ON THE AUTOMATION AND DIAGNOSIS OF VISUAL INTELLIGENCE

by
Chenxi Liu

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy

Baltimore, Maryland
July, 2020

Abstract

One of the ultimate goals of computer vision is to equip machines with visual intelligence: the ability to understand a scene at the level that is indistinguishable from human's. This not only requires detecting the 2D or 3D locations of objects, but also recognizing their semantic categories, or even higher level interactions. Thanks to decades of vision research as well as recent developments in deep learning, we are closer to this goal than ever. But to keep closing the gap, more research is needed on two themes. One, current models are still far from perfect, so we need a mechanism to keep proposing new, better models to improve performance. Two, while we are pushing for performance, it is also important to do careful analysis and diagnosis of existing models, to make sure we are indeed moving in the right direction.

In this dissertation, I study either of the two research themes for various steps in the visual intelligence pipeline. The first part of the dissertation focuses on category-level understanding of 2D images, which is arguably the most critical step in the visual intelligence pipeline as it bridges vision and language. The theme is on automating the process of model improvement: in particular, the architecture of neural networks. The second part extends the visual intelligence pipeline along the language side, and focuses on the more challenging language-level understanding of 2D images. The theme also shifts to diagnosis, by examining existing models, proposing interpretable models, or building diagnostic datasets. The third part continues in the diagnosis theme, this time extending along the vision side, focusing on how incorporating 3D scene knowledge may facilitate the evaluation of image recognition models.

Thesis Readers

Alan L. Yuille (Primary Advisor)
Bloomberg Distinguished Professor
Department of Cognitive Science & Computer Science
Johns Hopkins University

Gregory D. Hager
Mandell Bellmore Professor
Department of Computer Science
Johns Hopkins University

Fei-Fei Li
Sequoia Professor
Computer Science Department
Stanford University

Acknowledgments

First and foremost, I thank my advisor Alan Yuille. I got incredibly lucky, that my only Ph.D. offer is the one that suits me the best. Thank you for enabling the unique experience of spreading my Ph.D. study across two elite universities; for supporting and encouraging me in exploring interdisciplinary subjects, such as computational linguistics and cognitive science; for educating us with your passion and dedication by day; for sharing fun historical anecdotes and your British wit by night. Perhaps most importantly, you always have your students' interests at heart: this is felt and deeply appreciated.

I also thank my thesis committee: Greg Hager, for reminding me to lay a solid foundation in growing as a serious researcher, and Fei-Fei Li, for teaching me the importance of developing excellent research taste as well as zooming out to see the bigger picture.

There are more faculty members that have positively influenced my career, including several before graduate school. I thank Raquel Urtasun, Sanja Fidler, and Alexander Schwing, for introducing me to cutting-edge vision research during my summer at University of Toronto and for setting the standard high. I thank Jie Zhou, Greg Shakhnarovich, and Michael Maire, for guidance in continuing my research at Tsinghua and TTIC. I thank Fei Sha, for a unique seminar at UCLA that led to my first publication during graduate study. At Johns Hopkins, I thank Jason Eisner, formally for years of intellectual challenges, but really for just being awesome. I also thank Kyle Rawlins and Tal Linzen for serving on my GBO committee.

Thank you to my mentors and collaborators during internships: Zhe Lin, Xiaohui Shen,

Jimei Yang, Xin Lu at Adobe; Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Jia Li, Fei-Fei Li, Jonathan Huang, Kevin Murphy, Liang-Chieh Chen at Google; Piotr Dollár, Kaiming He, Ross Girshick, Saining Xie at Facebook. I was extremely fortunate to have worked with and learned from so many talented minds: it has been surreal.

I am thankful to all members of CCVL, past and present, for the countless discussions we had, meals we dined, and fun we shared. Among those who graduated from UCLA: Liang-Chieh Chen, Xianjie Chen, Zhou Ren, Fangting Xia, Junhua Mao, Peng Wang, Jianyu Wang, Xiaochen Lian, Alex Wong. Among those who helped form the lab at Johns Hopkins: Lingxi Xie, Jun Zhu, Vittal Premachandran, Ehsan Jahangiri, Xuan Dong, Wei Shen, Yan Wang, Adam Kortylewski, Yongyi Lu, Weichao Qiu, Zhuotun Zhu, Zhishuai Zhang, Siyuan Qiao, Cihang Xie, Yuyin Zhou, Chenxu Luo, Huiyu Wang, Qing Liu, Qi Chen, Fengze Liu, Yi Zhang, Hongru Zhu, Yingda Xia, Jieru Mei, Qihang Yu, Yingwei Li, Yixiao Zhang, Zhuowan Li, Zihao Xiao, Chenglin Yang, Yutong Bai, Angtian Wang, Chen Wei. I also sincerely thank my student advisees over the years, including Xiaohui Zeng, Yu-Siang Wang, Runtao Liu, Michelle Shu, Jiteng Mu: this dissertation would have been incomplete without them. The names are, of course, not exhaustive: the many more friends I made at schools, at conferences, at internships all helped shape who I am today.

I am thankful to the funding agencies that have supported my research, including fellowship from Google, Snap, NVIDIA.

Looking back on my pre-Ph.D. expectations, the amount of intellectual difficulties that I prepared for was about right; but what I underprepared for was the amount of psychological challenges. In this regard, I wholeheartedly thank my parents for being a constant source of support, and my girlfriend, Jingyu, for keeping me (virtual) company throughout the journey.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Visual Intelligence Pipeline

Computer vision is a large research field with a wide range of problems to solve, including tracking, correspondence, reconstruction, *etc*. However, personally I have always been most fascinated by what is called high-level vision: the set of problems that involves *semantic understanding* of the image or scene content. Only after such capabilities are achieved can machines seamlessly interact with humans, and may truly be called (visually) "intelligent".

The most basic form of visual intelligence is **category-level understanding** of **2D images**. Specifically, this requires correctly assigning a semantic category, such as "person" or "giraffe", to the entirety or part of an image. When a semantic category is assigned to the entire image, the task is often called image classification or image recognition. When a semantic category is assigned to each pixel in the image, the task is often called semantic segmentation or scene parsing. Thanks to the relative ease of collecting 2D images and annotating semantic labels (through the internet and crowdsourcing), these tasks have standard benchmark datasets, and are intensively studied. In fact, it was indeed category-level image understanding that ignited the current wave of deep learning [134].

But I argue that going from 2D images to semantic categories is only one step within the complete visual intelligence pipeline. For one, the semantic categories alone do not

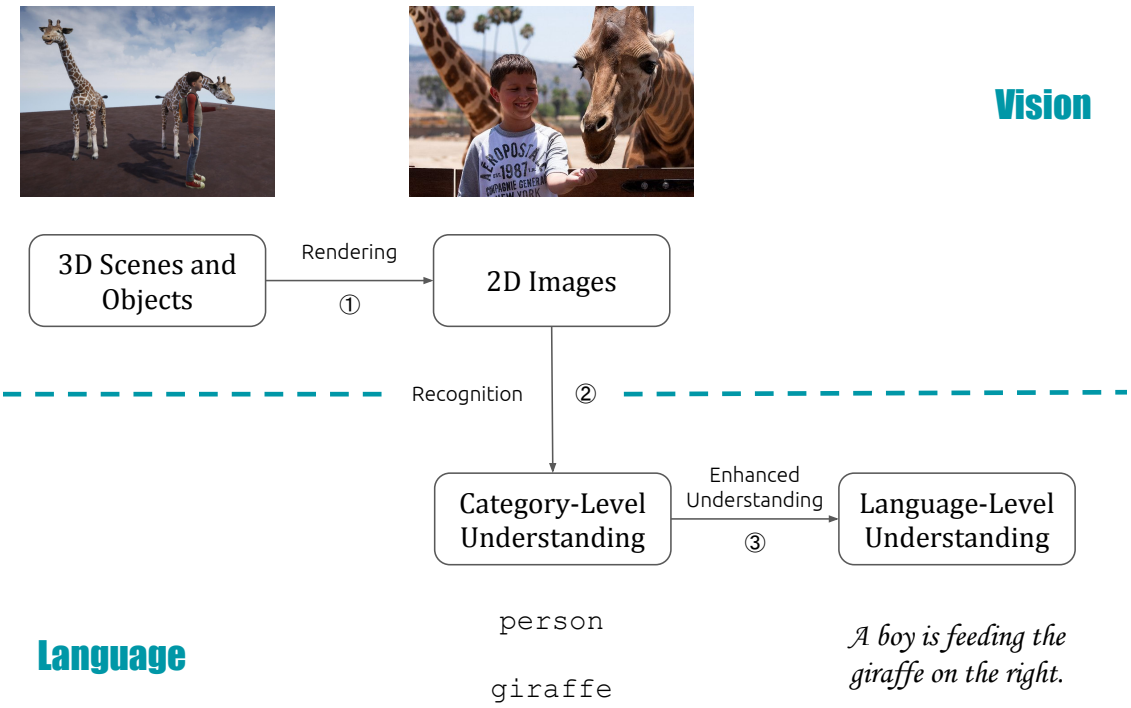


Figure 1.1. The visual intelligence pipeline considered in this dissertation. 3D scenes and objects create 2D images through rendering and projection. 2D images are then recognized in the form of semantic categories, or the more advanced level through natural language.

fully describe the image, such as the attributes of objects and the relations between objects. For example, the image in Figure 1.1 is much better understood and described by "A boy is feeding the giraffe on the right" than simply "person" and "giraffe". The word "boy" implies that the machine understood the age attribute of the "person" category, and the phrase "giraffe on the right" suggests that the machine has mastered the capability of referring to an unambiguous object. In this dissertation, I call this the **language-level understanding**, which is one step further than category-level.

In the other direction, though the 2D images already contain extremely rich information, it is important to bear in mind that the world is fundamentally in 3D. The 2D images are (merely) the results of rendering and projection of **3D scenes and objects**. Studying 2D image understanding is of course still valid and important, but as I will show later in this dissertation, having access to the 3D information will bring unique benefits, especially in

terms of diagnosis and evaluation.

Putting everything together, the visual intelligence pipeline considered in this dissertation starts from 3D scenes and objects, which produce 2D images, which then get recognized into semantic categories, or the more advanced level of language-level understanding. This pipeline is visualized in Figure 1.1. In the next section, I will describe the specific problems that I try to tackle, and how they situate within this big picture.

1.2 Outline

This dissertation consists of three parts, each corresponding to different steps in the visual intelligence pipeline with different emphasis.

1.2.1 Automating Structure Learning for 2D Image Recognition

Part I of the dissertation focuses on category-level understanding of 2D images, *i.e.* arrow ② in Figure 1.1. As aforementioned, this is the central topic of the recent wave of deep learning research. Specifically, significant performance jumps are usually attributed to better and better convolutional neural network *architectures* [89], [104], [134], [232], [245], [269]. However, these widely-used neural architectures designed by human experts only correspond to a tiny subset of all possible architectures. Can we utilize smart algorithms to let machines explore the uncharted territories in the neural architecture design space, and find architectures that outperform existing ones? This research direction, usually termed AutoML [109] or Neural Architecture Search (NAS) [297], is essentially the **automation** of the structure learning problem in the context of artificial neural networks. My research focuses on two aspects of this research direction: *acceleration* and *generalization*.

Early NAS methods are usually extremely costly. In order to find a high-quality neural architecture for 2D image recognition, the computation power required is around 2,000 GPU days. In Chapter 2 [154] I will describe an algorithm that brings this computation

cost down to around 150 GPU days while delivering an even better neural architecture. It was arguably the very first successful attempt at the *acceleration* aspect of modern NAS research.

In Chapter 3 [150], I will describe my efforts to *generalize* NAS beyond image classification problems to dense image prediction problems (such as semantic segmentation), where the goal is to assign a label to every pixel in the 2D image. This type of problems possesses unique challenges, in terms of designing search space and managing search cost. I will describe how these challenges are addressed, resulting in a new architecture for semantic segmentation that outperforms the previous state-of-the-art by a large margin.

Whether image classification or semantic segmentation, the structure learning was always conducted in a supervised fashion, in the sense that both the raw data and the annotated labels are required to find an architecture. In Chapter 4 [151] I defy this standard practice by *generalizing* NAS to the unsupervised setting. I will present the perhaps surprising finding that neural architectures discovered in an unsupervised fashion are just as good as those discovered in a supervised way.

1.2.2 Diagnosing Language-Level Understanding for 2D Images

Part II of this dissertation concerns language-level understanding of 2D images, *i.e*. arrow ② and ③ in Figure 1.1. Obviously, language-level understanding is a much more challenging problem than category-level, as it requires higher level, more detailed understanding. Although automation is still a valid topic, the emphasis here will shift to **diagnosis**. The general philosophy here is that in order to be convinced that the trained models have truly understood the image and the task, only looking at the output is not enough. Ideally, the model should be explainable, and should produce intermediate results that are meaningful and interpretable. I studied a variety of tasks that requires language-level understanding of 2D images.

In Chapter 5 [153] the task being considered is image captioning, *i.e*. generating a

sentence that describe the content of the image. Specifically, I analyze the attention mechanism in neural image captioning [272], which is a popular technique that not only generates the output (*i.e*. the words), but also provides where the model relies on when making this decision. There were qualitative studies that show these attention regions align with humans, but this phenomenon was never evaluated quantitatively. In this work, I define the concept of "attention correctness" and give a quantitative answer to the degree that these attention maps agree with humans.

In Chapter 6 [152] I consider a different task that is referring expression comprehension. The goal is to correctly select part of the image based on a referring expression, *e.g*., "the giraffe on the right". Instead of processing the image and the phrase separately, I will describe a model that facilitates exchange between these two modalities, which not only improves performance on various benchmark datasets, but also has the benefit of revealing the intermediate reasoning process, much like attention maps.

The study on referring expressions continues in Chapter 7 [158]. The benchmark datasets for referring expressions were collected with minimal restrictions, which means they are probably biased and do not exhaustively examine the model's reasoning ability. To overcome this limitation, I propose to instead rely on synthetic data, where the distribution over referring expressions can be explicitly controlled. Using this more balanced benchmark, I am able to thoroughly examine the strength and weakness of existing models, as well as proposing a new model with better modularity, explainability, and performance.

The main reason diagnosis is hard is because neural networks are usually considered "black boxes" and neural embeddings are hard to interpret. In Chapter 8 [256], I no longer use neural embeddings as the means to communicate between vision and language, but use the symbolic scene graph representation [114] instead. In order for the scene graph representation to be useful, one indispensable component is a parser that turns sentences into this graph representation. This chapter describes such a high-quality parser, and how it can be used to facilitate tasks such as image retrieval.

1.2.3 Diagnosing 2D Image Recognition with 3D Objects

Part III of the dissertation continues in the **diagnosis** theme. The focus here is how 3D information of the scene and objects may facilitate the evaluation and diagnosis of trained image recognition models. Therefore, this part corresponds to arrow ① and ② in Figure 1.1. Unfortunately, the 2D images collected from the internet usually do not come with the corresponding 3D information (for example, you do not expect the image in Figure 1.1 to come with the giraffe's precise 3D location, pose, and texture). For this reason, in this part I will rely on synthetic (but usually realistic) 3D environments to produce 2D images.

Researchers have long been aware that it is possible to slightly perturb pixel values to fail 2D image recognition [243]. However, these perturbations are usually high-frequency and difficult to encounter in real life. Instead, if 3D scenes and objects can be perturbed in a way that is much easier to achieve (for example, rotating the object slightly; increasing the lighting slightly) but still fail image recognition, then it is a much more worrisome phenomenon. In Chapter 9 [287], I show that such examples indeed exist, by incorporating rendering into the visual perception pipeline. This kind of diagnosis reveals previously underexplored weaknesses in trained image recognition models.

The finding above highlights a confusing paradox: trained image recognition models usually achieve very high performance on benchmark datasets (which may even surpass human performance), but at the same time, they have these blind spots that are fairly ubiquitous, which prove that these models are fragile and probably not ready to be deployed in sensitive applications. My answer to this paradox is that we need to consider an alternative *evaluation protocol*, which emphasizes worst-case scenarios over average-case scenarios. Using a similar experimental setup as Chapter 9, in Chapter 10 [229], I will describe a new evaluation protocol of such property, which can effectively prevent performance estimates from being overly optimistic.

Part I

Automating Structure Learning for 2D Image Recognition

Chapter 2

Progressive Neural Architecture Search

This chapter describes a method that accelerates the Neural Architecture Search process.

2.1 Introduction

There has been a lot of recent interest in automatically learning good neural net architectures. Some of this work is summarized in Section 2.2, but at a high level, current techniques usually fall into one of two categories: evolutionary algorithms (see e.g. [178], [210], [268]) or reinforcement learning (see e.g., [12], [21], [293], [297], [298]). When using evolutionary algorithms (EA), each neural network structure is encoded as a string, and random mutations and recombinations of the strings are performed during the search process; each string (model) is then trained and evaluated on a validation set, and the top performing models generate "children". When using reinforcement learning (RL), the agent performs a sequence of actions, which specifies the structure of the model; this model is then trained and its validation performance is returned as the reward, which is used to update the RNN controller. Although both EA and RL methods have been able to learn network structures that outperform manually designed architectures, they require significant computational resources. For example, the RL method in [298] trains and evaluates 20,000 neural networks across 500 P100 GPUs over 4 days.

In this chapter, we describe a method that is able to learn a CNN which matches

previous state of the art in terms of accuracy, while requiring 5 times fewer model evaluations during the architecture search. Our starting point is the structured search space proposed by [298], in which the search algorithm is tasked with searching for a good convolutional "cell", as opposed to a full CNN. A cell contains B "blocks", where a block is a combination operator (such as addition) applied to two inputs (tensors), each of which can be transformed (e.g., using convolution) before being combined. This cell structure is then stacked a certain number of times, depending on the size of the training set, and the desired running time of the final CNN (see Section 2.3 for details). This modular design also allows easy architecture transfer from one dataset to another, as we will show in experimental results.

We propose to use heuristic search to search the space of cell structures, starting with simple (shallow) models and progressing to complex ones, pruning out unpromising structures as we go. At iteration b of the algorithm, we have a set of K candidate cells (each of size b blocks), which we train and evaluate on a dataset of interest. Since this process is expensive, we also learn a model or surrogate function which can predict the performance of a structure without needing to train it. We expand the K candidates of size b into $K' \gg K$ children, each of size $b+1$. We apply our surrogate function to rank all of the K' children, pick the top K, and then train and evaluate them. We continue in this way until $b = B$, which is the maximum number of blocks we want to use in our cell. See Section 2.4 for details.

Our progressive (simple to complex) approach has several advantages over other techniques that directly search in the space of fully-specified structures. First, the simple structures train faster, so we get some initial results to train the surrogate quickly. Second, we only ask the surrogate to predict the quality of structures that are slightly different (larger) from the ones it has seen (c.f., trust-region methods). Third, we factorize the search space into a product of smaller search spaces, allowing us to potentially search models with many more blocks. In Section 2.5 we show that our approach is 5 times more efficient

than the RL method of [298] in terms of number of models evaluated, and 8 times faster in terms of total compute. We also show that the structures we discover achieve state of the art classification accuracies on CIFAR-10 and ImageNet.[1]

2.2 Related Work

This chapter is based on the "neural architecture search" (NAS) method proposed in [297], [298]. In the original paper [297], they use the REINFORCE algorithm [260] to estimate the parameters of a recurrent neural network (RNN), which represents a policy to generate a sequence of symbols (actions) specifying the structure of the CNN; the reward function is the classification accuracy on the validation set of a CNN generated from this sequence. [298] extended this by using a more structured search space, in which the CNN was defined in terms of a series of stacked "cells". (They also replaced REINFORCE with proximal policy optimization (PPO) [218].) This method was able to learn CNNs which outperformed almost all previous methods in terms of accuracy vs speed on image classification (using CIFAR-10 [133] and ImageNet [47]) and object detection (using COCO [148]).

There are several other papers that use RL to learn network structures. [293] use the same model search space as NAS, but replace policy gradient with Q-learning. [12] also use Q-learning, but without exploiting cell structure. [21] use policy gradient to train an RNN, but the actions are now to widen an existing layer, or to deepen the network by adding an extra layer. This requires specifying an initial model and then gradually learning how to transform it. The same approach, of applying "network morphisms" to modify a network, was used in [58], but in the context of hill climbing search, rather than RL. [202] use parameter sharing among child models to substantially accelerate the search

[1] The code and checkpoint for the PNAS model trained on ImageNet can be downloaded from the TensorFlow models repository at `http://github.com/tensorflow/models/`. Also see `https://github.com/chenxi116/PNASNet.TF` and `https://github.com/chenxi116/PNASNet.pytorch` for author's reimplementation.

process.

An alternative to RL is to use evolutionary algorithms (EA; "neuro-evolution" [239]). Early work (e.g., [240]) used EA to learn both the structure and the parameters of the network, but more recent methods, such as [156], [178], [208], [210], [268], just use EA to search the structures, and use SGD to estimate the parameters.

RL and EA are local search methods that search through the space of fully-specified graph structures. An alternative approach, which we adopt, is to use heuristic search, in which we search through the space of structures in a progressive way, from simple to complex. There are several pieces of prior work that explore this approach. [187] use Monte Carlo Tree Search (MCTS), but at each node in the search tree, it uses random selection to choose which branch to expand, which is very inefficient. Sequential Model Based Optimization (SMBO) [108] improves on MCTS by learning a predictive model, which can be used to decide which nodes to expand. This technique has been applied to neural net structure search in [187], but they used a flat CNN search space, rather than our hierarchical cell-based space. Consequently, their resulting CNNs do not perform very well. Other related works include [176], who focus on MLP rather than CNNs; [240], who used an incremental approach in the context of evolutionary algorithms; [297] who used a schedule of increasing number of layers; and [83] who search through the space of latent factor models specified by a grammar. Finally, [43], [103] grow CNNs sequentially using boosting.

Several other papers learn a surrogate function to predict the performance of a candidate structure, either "zero shot" (without training it) (see e.g., [17]), or after training it for a small number of epochs and extrapolating the learning curve (see e.g., [13], [51]). However, most of these methods have been applied to fixed sized structures, and would not work with our progressive search approach.

2.3 Architecture Search Space

In this section we describe the neural network architecture search space used in our work. We build on the hierarchical approach proposed in [298], in which we first learn a cell structure, and then stack this cell a desired number of times, in order to create the final CNN.

2.3.1 Cell Topologies

A cell is a fully convolutional network that maps an $H \times W \times F$ tensor to another $H' \times W' \times F'$ tensor. If we use stride 1 convolution, then $H' = H$ and $W' = W$; if we use stride 2, then $H' = H/2$ and $W' = W/2$. We employ a common heuristic to double the number of filters (feature maps) whenever the spatial activation is halved, so $F' = F$ for stride 1, and $F' = 2F$ for stride 2.

The cell can be represented by a DAG consisting of B blocks. Each block is a mapping from 2 input tensors to 1 output tensor. We can specify a block b in a cell c as a 5-tuple, (I_1, I_2, O_1, O_2, C), where $I_1, I_2 \in \mathcal{I}_b$ specifies the inputs to the block, $O_1, O_2 \in \mathcal{O}$ specifies the operation to apply to input I_i, and $C \in \mathcal{C}$ specifies how to combine O_1 and O_2 to generate the feature map (tensor) corresponding to the output of this block, which we denote by H_b^c.

The set of possible inputs, $\mathcal{I}_b$, is the set of all previous blocks in this cell, $\{H_1^c, \ldots, H_{b-1}^c\}$, plus the output of the previous cell, H_B^{c-1}, plus the output of the previous-previous cell, H_B^{c-2}.

The operator space $\mathcal{O}$ is the following set of 8 functions, each of which operates on a single tensor[2]:

[2] The depthwise-separable convolutions are in fact two repetitions of ReLU-SepConv-BatchNorm; 1x1 convolutions are also inserted when tensor sizes mismatch.

- 3x3 depthwise-separable convolution
- 5x5 depthwise-separable convolution
- 7x7 depthwise-separable convolution
- 1x7 followed by 7x1 convolution
- identity
- 3x3 average pooling
- 3x3 max pooling
- 3x3 dilated convolution

This is less than the 13 operators used in [298], since we removed the ones that their RL method discovered were never used.

For the space of possible combination operators $\mathcal{C}$, [298] considerd both elementwise addition and concatenation. However, they discovered that the RL method never chose to use concatenation, so to reduce our search space, we always use addition as the combination operator. Thus in our work, a block can be specified by a 4-tuple.

We now quantify the size of the search space to highlight the magnitude of the search problem. Let the space of possible structures for the b'th block be $\mathcal{B}_b$; this has size $|\mathcal{B}_b| = |\mathcal{I}_b|^2 \times |\mathcal{O}|^2 \times |\mathcal{C}|$, where $|\mathcal{I}_b| = (2 + b - 1)$, $|\mathcal{O}| = 8$ and $|\mathcal{C}| = 1$. For $b = 1$, we have $\mathcal{I}_1 = \{H_B^{c-1}, H_B^{c-2}\}$, which are the final outputs of the previous two cells, so there are $|\mathcal{B}_1| = 256$ possible block structures.

If we allow cells of up to $B = 5$ blocks, the total number of cell structures is given by $|\mathcal{B}_{1:5}| = 2^2 \times 8^2 \times 3^2 \times 8^2 \times 4^2 \times 8^2 \times 5^2 \times 8^2 \times 6^2 \times 8^2 = 5.6 \times 10^{14}$. However, there are certain symmetries in this space that allow us to prune it to a more reasonable size. For example, there are only 136 unique cells composed of 1 block. The total number of unique cells is $\sim 10^{12}$. This is much smaller than the search space used in [298], which has size 10^{28}, but it is still an extremely large space to search, and requires efficient optimization methods.

2.3.2 From Cell to CNN

To evaluate a cell, we have to convert it into a CNN. To do this, we stack a predefined number of copies of the basic cell (with the same structure, but untied weights), using

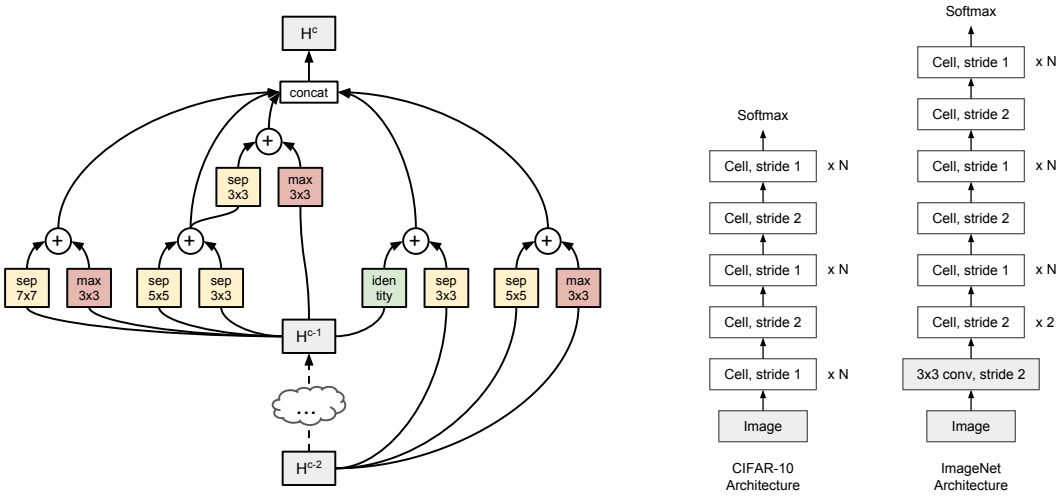


Figure 2.1. *Left*: The best cell structure found by our Progressive Neural Architecture Search, consisting of 5 blocks. *Right*: We employ a similar strategy as [298] when constructing CNNs from cells on CIFAR-10 and ImageNet. Note that we learn a single cell type instead of distinguishing between Normal and Reduction cell.

either stride 1 or stride 2, as shown in Figure 2.1 (right). The number of stride-1 cells between stride-2 cells is then adjusted accordingly with up to N number of repeats. At the top of the network, we use global average pooling, followed by a softmax classification layer. We then train the stacked model on the relevant dataset.

In the case of CIFAR-10, we use 32×32 images. In the case of ImageNet, we consider two settings, one with high resolution images of size 331×331, and one with smaller images of size 224×224. The latter results in less accurate models, but they are faster. For ImageNet, we also add an initial 3×3 convolutional filter layer with stride 2 at the start of the network, to further reduce the cost.

The overall CNN construction process is identical to [298], except we only use one cell type (we do not distinguish between Normal and Reduction cells, but instead emulate a Reduction cell by using a Normal cell with stride 2), and the cell search space is slightly smaller (since we use fewer operators and combiners).

Algorithm 2.1: Progressive Neural Architecture Search (PNAS).

Inputs: B (max num blocks), E (max num epochs), F (num filters in first layer), K (beam size), N (num times to unroll cell), trainSet, valSet.
$\mathcal{S}_1 = \mathcal{B}_1$ // *Set of candidate structures with one block*
$\mathcal{M}_1$ = cell-to-CNN($\mathcal{S}_1$, N, F) // *Construct CNNs from cell specifications*
$\mathcal{C}_1$ = train-CNN($\mathcal{M}_1$, E, trainSet) // *Train proxy CNNs*
$\mathcal{A}_1$ = eval-CNN($\mathcal{C}_1$, valSet) // *Validation accuracies*
π = fit($\mathcal{S}_1$, $\mathcal{A}_1$) // *Train the reward predictor from scratch*
for $b = 2 : B$ **do**
  $\mathcal{S}'_b$ = expand-cell($\mathcal{S}_{b-1}$) // *Expand current candidate cells by one more block*
  $\hat{\mathcal{A}}'_b$ = predict($\mathcal{S}'_b$, π) // *Predict accuracies using reward predictor*
  $\mathcal{S}_b$ = top-K($\mathcal{S}'_b$, $\hat{\mathcal{A}}'_b$, K) // *Most promising cells according to prediction*
  $\mathcal{M}_b$ = cell-to-CNN($\mathcal{S}_b$, N, F)
  $\mathcal{C}_b$ = train-CNN($\mathcal{M}_b$, E, trainSet)
  $\mathcal{A}_b$ = eval-CNN($\mathcal{C}_b$, valSet)
  π = update-predictor($\mathcal{S}_b$, $\mathcal{A}_b$, π) // *Finetune reward predictor with new data*
end for
Return top-K($\mathcal{S}_B$, $\mathcal{A}_B$, 1)

2.4 Method

2.4.1 Progressive Neural Architecture Search

Many previous approaches directly search in the space of full cells, or worse, full CNNs. For example, NAS uses a 50-step RNN[3] as a controller to generate cell specifications. In [268] a fixed-length binary string encoding of CNN architecture is defined and used in model evolution/mutation. While this is a more direct approach, we argue that it is difficult to directly navigate in an exponentially large search space, especially at the beginning where there is no knowledge of what makes a good model.

As an alternative, we propose to search the space in a progressive order, simplest models first. In particular, we start by constructing all possible cell structures from $\mathcal{B}_1$ (i.e., composed of 1 block), and add them to a queue. We train and evaluate all the models in the queue (in parallel), and then expand each one by adding all of the possible block structures from $\mathcal{B}_2$; this gives us a set of $|\mathcal{B}_1| \times |\mathcal{B}_2| = 256 \times 576 = 147,456$ candidate cells

[3] 5 symbols per block, times 5 blocks, times 2 for Normal and Reduction cells.

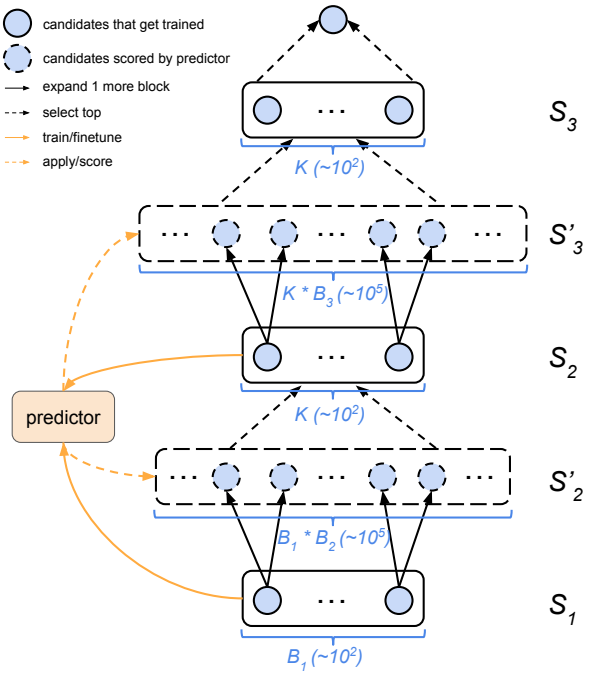


Figure 2.2. Illustration of the PNAS search procedure when the maximum number of blocks is $B = 3$. Here $\mathcal{S}_b$ represents the set of candidate cells with b blocks. We start by considering all cells with 1 block, $\mathcal{S}_1 = \mathcal{B}_1$; we train and evaluate all of these cells, and update the predictor. At iteration 2, we expand each of the cells in $\mathcal{S}_1$ to get all cells with 2 blocks, $\mathcal{S}'_2 = \mathcal{B}_{1:2}$; we predict their scores, pick the top K to get $\mathcal{S}_2$, train and evaluate them, and update the predictor. At iteration 3, we expand each of the cells in $\mathcal{S}_2$, to get a subset of cells with 3 blocks, $\mathcal{S}'_3 \subseteq \mathcal{B}_{1:3}$; we predict their scores, pick the top K to get $\mathcal{S}_3$, train and evaluate them, and return the winner. $B_b = |\mathcal{B}_b|$ is the number of possible blocks at level b and K is the beam size (number of models we train and evaluate per level of the search tree).

of depth 2. Since we cannot afford to train and evaluate all of these child networks, we refer to a learned predictor function (described in Section 2.4.2); it is trained based on the measured performance of the cells we have visited so far. (Our predictor takes negligible time to train and apply.) We then use the predictor to evaluate all the candidate cells, and pick the K most promising ones. We add these to the queue, and repeat the process, until we find cells with a sufficient number B of blocks. See Algorithm 2.1 for the pseudocode, and Figure 2.2 for an illustration.

2.4.2 Performance Prediction with Surrogate Model

As explained above, we need a mechanism to predict the final performance of a cell before we actually train it. There are at least three desired properties of such a predictor:

- *Handle variable-sized inputs*: We need the predictor to work for variable-length input strings. In particular, it should be able to predict the performance of any cell with

$b + 1$ blocks, even if it has only been trained on cells with up to b blocks.

- *Correlated with true performance*: we do not necessarily need to achieve low mean squared error, but we do want the predictor to rank models in roughly the same order as their true performance values.

- *Sample efficiency*: We want to train and evaluate as few cells as possible, which means the training data for the predictor will be scarce.

The requirement that the predictor be able to handle variable-sized strings immediately suggests the use of an RNN, and indeed this is one of the methods we try. In particular, we use an LSTM that reads a sequence of length $4b$ (representing I_1, I_2, O_1 and O_2 for each block), and the input at each step is a one-hot vector of size $|\mathcal{I}_b|$ or $|\mathcal{O}|$, followed by embedding lookup. We use a shared embedding of dimension D for the tokens $I_1, I_2 \in \mathcal{I}$, and another shared embedding for $O_1, O_2 \in \mathcal{O}$. The final LSTM hidden state goes through a fully-connected layer and sigmoid to regress the validation accuracy. We also try a simpler MLP baseline in which we convert the cell to a fixed length vector as follows: we embed each token into an D-dimensional vector, concatenate the embeddings for each block to get an $4D$-dimensional vector, and then average over blocks. Both models are trained using L_1 loss.

When training the predictor, one approach is to update the parameters of the predictor using the new data using a few steps of SGD. However, since the sample size is very small, we fit an ensemble of 5 predictors, each fit (from scratch) to $4/5$ of all the data available at each step of the search process. We observed empirically that this reduced the variance of the predictions.

In the future, we plan to investigate other kinds of predictors, such as Gaussian processes with string kernels (see e.g., [11]), which may be more sample efficient to train and produce predictions with uncertainty estimates.

2.5 Experiments and Results

2.5.1 Experimental Details

Our experimental setting follows [298]. In particular, we conduct most of our experiments on CIFAR-10 [133]. CIFAR-10 has 50,000 training images and 10,000 test images. We use 5000 images from the training set as a validation set. All images are whitened, and 32×32 patches are cropped from images upsampled to 40×40. Random horizontal flip is also used. After finding a good model on CIFAR-10, we evaluate its quality on ImageNet classification in Section 2.5.5.

For the MLP accuracy predictor, the embedding size is 100, and we use 2 fully connected layers, each with 100 hidden units. For the RNN accuracy predictor, we use an LSTM, and the hidden state size and embedding size are both 100. The embeddings use uniform initialization in range [-0.1, 0.1]. The bias term in the final fully connected layer is initialized to 1.8 (0.86 after sigmoid) to account for the mean observed accuracy of all $b = 1$ models. We use the Adam optimizer [125] with learning rate 0.01 for the $b = 1$ level and 0.002 for all following levels.

Our training procedure for the CNNs follows the one used in [298]. During the search we evaluate $K = 256$ networks at each stage (136 for stage 1, since there are only 136 unique cells with 1 block), we use a maximum cell depth of $B = 5$ blocks, we use $F = 24$ filters in the first convolutional cell, we unroll the cells for $N = 2$ times, and each child network is trained for 20 epochs using initial learning rate of 0.01 with cosine decay [162].

2.5.2 Performance of the Surrogate Predictors

In this section, we compare the performance of different surrogate predictors. Note that at step b of PNAS, we train the predictor on the observed performance of cells with up to b blocks, but we apply it to cells with $b + 1$ blocks. We therefore consider predictive accuracy both for cells with sizes that have been seen before (but which have not been

trained on), and for cells which are one block larger than the training data.

More precisely, let $\mathcal{U}_{b,1:R}$ be a set of randomly chosen cells with b blocks, where $R = 10,000$. (For $b = 1$, there are only 136 unique cells.) We convert each of these to CNNs, and train them for $E = 20$ epochs. (Thus in total we train $\sim (B-1) \times R = 40,000$ models for 20 epochs each.) We now use this random dataset to evaluate the performance of the predictors using the pseudocode in Algorithm 2.2, where $A(\mathcal{H})$ returns the true validation set accuracies of the models in some set $\mathcal{H}$. In particular, for each size $b = 1 : B$, and for each trial $t = 1 : T$ (we use $T = 20$), we do the following: randomly select $K = 256$ models (each of size b) from $\mathcal{U}_{b,1:R}$ to generate a training set $\mathcal{S}_{b,t,1:K}$; fit the predictor on the training set; evaluate the predictor on the training set; and finally evaluate the predictor on the set of all unseen random models of size $b + 1$.

The top row of Figure 2.3 shows a scatterplot of the true accuracies of the models in the training sets, $A(\mathcal{S}_{b,1:T,1:K})$, vs the predicted accuracies, $\hat{A}_{b,1:T,1:K}$ (so there are $T \times K = 20 \times 256 = 5120$ points in each plot, at least for $b > 1$). The bottom row plots the true accuracies on the set of larger models, $A(\mathcal{U}_{b+1,1:R})$, vs the predicted accuracies $\tilde{A}_{b+1,1:R}$ (so there are $R = 10$K points in each plot). We see that the predictor performs well on models from the training set, but not so well when predicting larger models. However, performance does increase as the predictor is trained on more (and larger) cells.

Figure 2.3 shows the results using an ensemble of MLPs. The scatter plots for the other predictors look similar. We can summarize each scatterplot using the Spearman rank

Algorithm 2.2: Evaluating performance of a predictor on a random dataset.

for $b = 1 : B - 1$ **do**
  for $t = 1 : T$ **do**
    $\mathcal{S}_{b,t,1:K}$ = random sample of K models from $\mathcal{U}_{b,1:R}$
    $\pi_{b,t}$ = fit($\mathcal{S}_{b,t,1:K}$, $A(\mathcal{S}_{b,t,1:K})$) // *Train or finetune predictor*
    $\hat{A}_{b,t,1:K}$ = predict($\pi_{b,t}$, $\mathcal{S}_{b,t,1:K}$) // *Predict on same b*
    $\tilde{A}_{b+1,t,1:R}$ = predict($\pi_{b,t}$, $\mathcal{U}_{b+1,1:R}$) // *Predict on next b*
  end for
end for

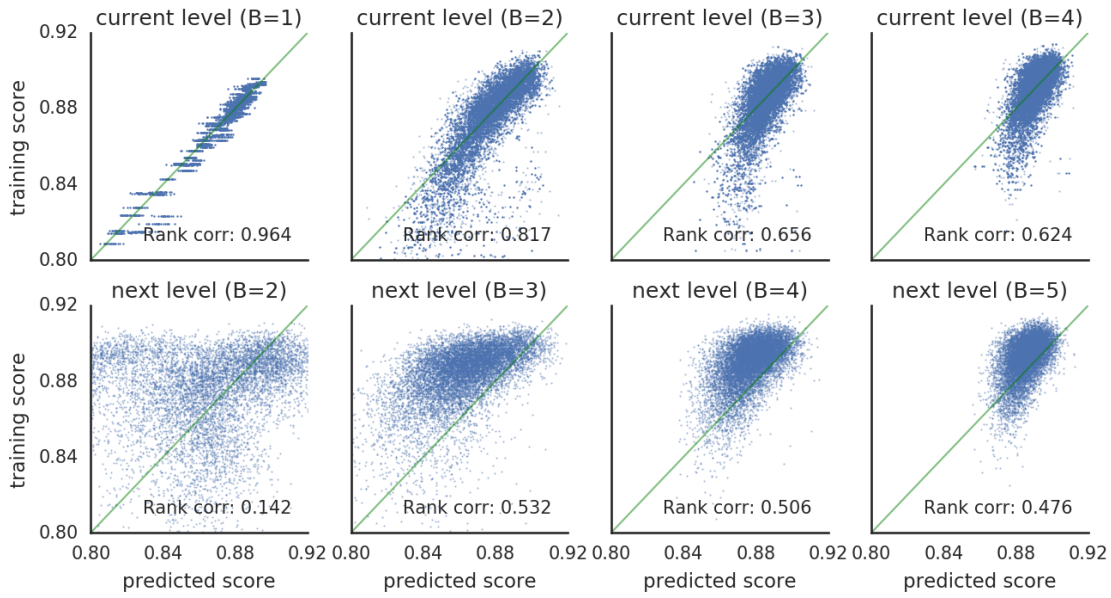


Figure 2.3. Accuracy of MLP-ensemble predictor. Top row: true vs predicted accuracies on models from the training set over different trials. Bottom row: true vs predicted accuracies on models from the set of all unseen larger models. Denoted is the mean rank correlation from individual trials.

correlation coefficient. Let $\hat{\rho}_b$ = rank-correlation($\hat{A}_{b,1:T,1:K}$, $A(\mathcal{S}_{b,1:T,1:K})$) and $\tilde{\rho}_{b+1}$ = rank-correlation($\tilde{A}_{b+1,1:R}$, $A(\mathcal{U}_{b+1,1:R})$). Table 2.1 summarizes these statistics across different levels. We see that for predicting the training set, the RNN does better than the MLP, but for predicting the performance on unseen larger models (which is the setting we care about in practice), the MLP seems to do slightly better. This will be corroborated by our end-to-end test in Section 2.5.3, and is likely due to overfitting. We also see that for the extrapolation task, ensembling seems to help.

2.5.3 Search Efficiency

In this section, we compare the efficiency of PNAS to two other methods: random search and the NAS method. To perform the comparison, we run PNAS for $B = 5$, and at each iteration b, we record the set $\mathcal{S}_b$ of $K = 256$ models of size b that it picks, and evaluate them on the CIFAR-10 validation set (after training for 20 epochs each). We then compute the validation accuracy of the top M models for $M \in \{1, 5, 25\}$. To capture the

Method	$b=1$		$b=2$		$b=3$		$b=4$	
	$\hat{\rho}_1$	$\tilde{\rho}_2$	$\hat{\rho}_2$	$\tilde{\rho}_3$	$\hat{\rho}_3$	$\tilde{\rho}_4$	$\hat{\rho}_4$	$\tilde{\rho}_5$
MLP	0.938	0.113	0.857	0.450	0.714	0.469	0.641	0.444
RNN	0.970	**0.198**	**0.996**	0.424	0.693	0.401	**0.787**	0.413
MLP-ensemble	**0.975**	0.164	0.786	**0.532**	0.634	**0.504**	0.645	**0.468**
RNN-ensemble	0.972	0.164	0.906	0.418	**0.801**	0.465	0.579	0.424

Table 2.1. Spearman rank correlations of different predictors on the training set, $\hat{\rho}_b$, and when extrapolating to unseen larger models, $\tilde{\rho}_{b+1}$. See text for details.

variance in performance of a given model due to randomness of the parameter initialization and optimization procedure, we repeat this process 5 times. We plot the mean and standard error of this statistic in Figure 2.4. We see that the mean performance of the top $M \in \{1, 5, 25\}$ models steadily increases, as we search for larger models. Furthermore, performance is better when using an MLP-ensemble (shown in Figure 2.4) instead of an RNN-ensemble (see Appendix A), which is consistent with Table 2.1.

For our random search baseline, we uniformly sample 6000 cells of size $B = 5$ blocks from the random set of models $\mathcal{U}_{5,1:R}$ described in Section 2.5.2. Figure 2.4 shows that PNAS significantly outperforms this baseline.

Finally, we compare to NAS. Each trial sequentially searches 6000 cells of size $B = 5$ blocks. At each iteration t, we define H_t to be the set of all cells visited so far by the RL agent. We compute the validation accuracy of the top M models in H_t, and plot the mean and standard error of this statistic in Figure 2.4. We see that the mean performance steadily increases, but at a slower rate than PNAS.

To quantify the speedup factor compared to NAS, we compute the number of models that are trained and evaluated until the mean performance of PNAS and NAS are equal (note that PNAS produces models of size B after evaluating $|\mathcal{B}_1| + (B-1) \times K$ models, which is 1160 for $B = 5$). The results are shown in Table 2.2. We see that PNAS is up to 5 times faster in terms of the number of models it trains and evaluates.

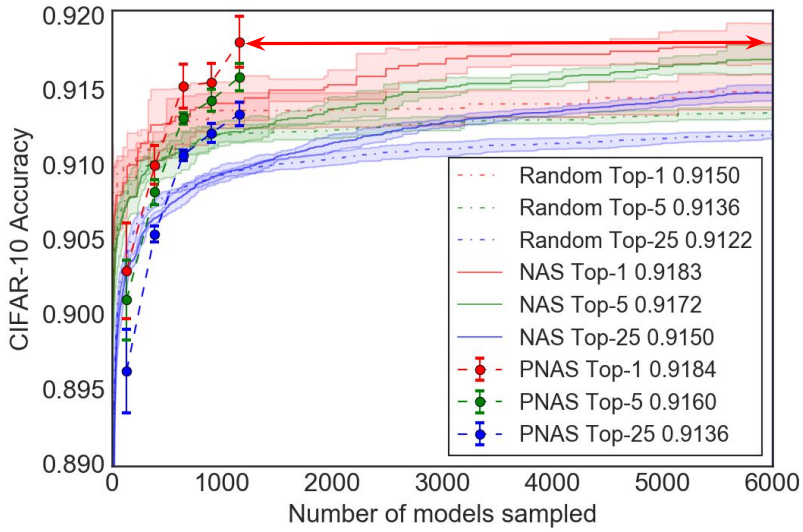


Figure 2.4. Comparing the relative efficiency of NAS, PNAS and random search under the same search space. We plot mean accuracy (across 5 trials) on CIFAR-10 validation set of the top M models, for $M \in \{1, 5, 25\}$, found by each method vs number of models which are trained and evaluated. Each model is trained for 20 epochs. Error bars and the colored regions denote standard deviation of the mean.

Comparing the number of models explored during architecture search is one measure of efficiency. However, some methods, such as NAS, employ a secondary reranking stage to determine the best model; PNAS does not perform a reranking stage but uses the top model from the search directly. A more fair comparison is therefore to count the total number of examples processed through SGD throughout the search. Let M_1 be the number of models trained during search, and let E_1 be the number of examples used to train each model.[4] The total number of examples is therefore $M_1 E_1$. However, for methods with the additional reranking stage, the top M_2 models from the search procedure are trained using E_2 examples each, before returning the best. This results in a total cost of $M_1 E_1 + M_2 E_2$. For NAS and PNAS, $E_1 = 900\text{K}$ for NAS and PNAS since they use 20 epochs on a training set of size 45K. The number of models searched to achieve equal top-1

[4] The number of examples is equal to the number of SGD steps times the batch size. Alternatively, it can be measured in terms of number of epoch (passes through the data), but since different papers use different sized training sets, we avoid this measure. In either case, we assume the number of examples is the same for every model, since none of the methods we evaluate use early stopping.

B	Top	Accuracy	# PNAS	# NAS	Speedup (# models)	Speedup (# examples)
5	1	0.9183	1160	5808	5.0	8.2
5	5	0.9161	1160	4100	3.5	6.8
5	25	0.9136	1160	3654	3.2	6.4

Table 2.2. Relative efficiency of PNAS (using MLP-ensemble predictor) and NAS under the same search space. B is the size of the cell, "Top" is the number of top models we pick, "Accuracy" is their average validation accuracy, "# PNAS" is the number of models evaluated by PNAS, "# NAS" is the number of models evaluated by NAS to achieve the desired accuracy. Speedup measured by number of examples is greater than speedup in terms of number of models, because NAS has an additional reranking stage, that trains the top 250 models for 300 epochs each before picking the best one.

accuracy is $M_1 = 1160$ for PNAS and $M_1 = 5808$ for NAS. For the second stage, NAS trains the top $M_2 = 250$ models for $E_2 = 300$ epochs before picking the best.[5] Thus we see that PNAS is about 8 times faster than NAS when taking into account the total cost.

2.5.4 Results on CIFAR-10 Image Classification

We now discuss the performance of our final model, and compare it to the results of other methods in the literature. Let PNASNet-5 denote the best CNN we discovered on CIFAR using PNAS, also visualized in Figure 2.1 (left). After we have selected the cell structure, we try various N and F values such that the number of model parameters is around 3M, train them each for 300 epochs using initial learning rate of 0.025 with cosine decay, and pick the best combination based on the validation set. Using this best combination of N and F, we train it for 600 epochs on the union of training set and validation set. During training we also used auxiliary classifier located at 2/3 of the maximum depth weighted by 0.4, and drop each path with probability 0.4 for regularization.

The results are shown in Table 2.3. We see that PNAS can find a model with the same

[5] This additional stage is quite important for NAS, as the NASNet-A cell was originally ranked 70th among the top 250.

Model	B	N	F	Error	Params	M_1	E_1	M_2	E_2	Cost
NASNet-A [298]	5	6	32	3.41	3.3M	20000	0.9M	250	13.5M	21.4-29.3B
NASNet-B [298]	5	4	N/A	3.73	2.6M	20000	0.9M	250	13.5M	21.4-29.3B
NASNet-C [298]	5	4	N/A	3.59	3.1M	20000	0.9M	250	13.5M	21.4-29.3B
Hier-EA [156]	5	2	64	3.75±0.12	15.7M	7000	5.12M	0	0	35.8B[6]
AmoebaNet-B [208]	5	6	36	3.37±0.04	2.8M	27000	2.25M	100	27M	63.5B[7]
AmoebaNet-A [208]	5	6	36	3.34±0.06	3.2M	20000	1.13M	100	27M	25.2B[8]
PNASNet-5	5	3	48	3.41±0.09	3.2M	1160	0.9M	0	0	1.0B

Table 2.3. Performance of different CNNs on CIFAR test set. All model comparisons employ a comparable number of parameters and exclude cutout data augmentation [49]. "Error" is the top-1 misclassification rate on the CIFAR-10 test set. (Error rates have the form $\mu \pm \sigma$, where μ is the average over multiple trials and σ is the standard deviation. In PNAS we use 15 trials.) "Params" is the number of model parameters. "Cost" is the total number of examples processed through SGD ($M_1E_1 + M_2E_2$) before the architecture search terminates. The number of filters F for NASNet-{B, C} cannot be determined (hence N/A), and the actual E_1, E_2 may be larger than the values in this table (hence the range in cost), according to the original authors.

accuracy as NAS, but using 21 times less compute. PNAS also outperforms the Hierarchical EA method of [156], while using 36 times less compute. Though the the EA method called "AmoebaNets" [208] currently give the highest accuracies (at the time of writing), it also requires the most compute, taking 63 times more resources than PNAS. However, these comparisons must be taken with a grain of salt, since the methods are searching through different spaces. By contrast, in Section 2.5.3, we fix the search space for NAS and PNAS, to make the speedup comparison fair.

[6] In Hierarchical EA, the search phase trains 7K models (each for 4 times to reduce variance) for 5000 steps of batch size 256. Thus, the total computational cost is 7K $\times$ 5000 $\times$ 256 $\times$ 4 = 35.8B.

[7] The total computational cost for AmoebaNet consists of an architecture search and a reranking phase. The architecture search phase trains over 27K models each for 50 epochs. Each epoch consists of 45K examples. The reranking phase searches over 100 models each trained for 600 epochs. Thus, the architecture search is 27K $\times$ 50 $\times$ 45K = 60.8B examples. The reranking phase consists of 100 $\times$ 600 $\times$ 45K = 2.7B examples. The total computational cost is 60.8B + 2.7B = 63.5B.

[8] The search phase trains 20K models each for 25 epochs. The rest of the computation is the same as AmoebaNet-B.

Model	Params	Mult-Adds	Top-1	Top-5
MobileNet-224 [95]	4.2M	569M	70.6	89.5
ShuffleNet (2x) [290]	5M	524M	70.9	89.8
NASNet-A ($N = 4$, $F = 44$) [298]	5.3M	564M	74.0	91.6
AmoebaNet-B ($N = 3$, $F = 62$) [208]	5.3M	555M	74.0	91.5
AmoebaNet-A ($N = 4$, $F = 50$) [208]	5.1M	555M	74.5	92.0
AmoebaNet-C ($N = 4$, $F = 50$) [208]	6.4M	570M	75.7	92.4
PNASNet-5 ($N = 3$, $F = 54$)	5.1M	588M	74.2	91.9

Table 2.4. ImageNet classification results in the *Mobile* setting.

2.5.5 Results on ImageNet Image Classification

We further demonstrate the usefulness of our learned cell by applying it to ImageNet classification. Our experiments reveal that CIFAR accuracy and ImageNet accuracy are strongly correlated ($\rho = 0.727$; see Appendix A).

To compare the performance of PNASNet-5 to the results in other papers, we conduct experiments under two settings:

- *Mobile*: Here we restrain the representation power of the CNN. Input image size is 224×224, and the number of multiply-add operations is under 600M.
- *Large*: Here we compare PNASNet-5 against the state-of-the-art models on ImageNet. Input image size is 331×331.

In both experiments we use RMSProp optimizer, label smoothing of 0.1, auxiliary classifier located at 2/3 of the maximum depth weighted by 0.4, weight decay of 4e-5, and dropout of 0.5 in the final softmax layer. In the *Mobile* setting, we use distributed synchronous SGD with 50 P100 workers. On each worker, batch size is 32, initial learning rate is 0.04, and is decayed every 2.2 epochs with rate 0.97. In the *Large* setting, we use 100 P100 workers. On each worker, batch size is 16, initial learning rate is 0.015, and is decayed every 2.4 epochs with rate 0.97. During training, we drop each path with probability 0.4.

Model	Image Size	Params	Mult-Adds	Top-1	Top-5
ResNeXt-101 (64x4d) [269]	320×320	83.6M	31.5B	80.9	95.6
PolyNet [291]	331×331	92M	34.7B	81.3	95.8
Dual-Path-Net-131 [37]	320×320	79.5M	32.0B	81.5	95.8
Squeeze-Excite-Net [96]	320×320	145.8M	42.3B	82.7	96.2
NASNet-A ($N = 6$, $F = 168$) [298]	331×331	88.9M	23.8B	82.7	96.2
AmoebaNet-B ($N = 6$, $F = 190$) [208]	331×331	84.0M	22.3B	82.3	96.1
AmoebaNet-A ($N = 6$, $F = 190$) [208]	331×331	86.7M	23.1B	82.8	96.1
AmoebaNet-C ($N = 6$, $F = 228$) [208]	331×331	155.3M	41.1B	83.1	96.3
PNASNet-5 ($N = 4$, $F = 216$)	331×331	86.1M	25.0B	82.9	96.2

Table 2.5. ImageNet classification results in the *Large* setting.

The results of the *Mobile* setting are summarized in Table 2.4. PNASNet-5 achieves slightly better performance than NASNet-A (74.2% top-1 accuracy for PNAS vs 74.0% for NASNet-A). Both methods significantly surpass the previous state-of-the-art, which includes the manually designed MobileNet [95] (70.6%) and ShuffleNet [290] (70.9%). AmoebaNet-C performs the best, but note that this is a different model than their best-performing CIFAR-10 model. Table 2.5 shows that under the *Large* setting, PNASNet-5 achieves higher performance (82.9% top-1; 96.2% top-5) than previous state-of-the-art approaches, including SENet [96], NASNet-A, and AmoebaNets under the same model capacity.

2.6 Discussion and Future Work

The main contribution of this work is to show how we can accelerate the search for good CNN structures by using progressive search through the space of increasingly complex graphs, combined with a learned prediction function to efficiently identify the most promising models to explore. The resulting models achieve the same level of performance as previous work but with a fraction of the computational cost.

There are many possible directions for future work, including: the use of better surro-

gate predictors, such as Gaussian processes with string kernels; the use of model-based early stopping, such as [13], so we can stop the training of "unpromising" models before reaching E_1 epochs; the use of "warm starting", to initialize the training of a larger $b+1$-sized model from its smaller parent; the use of Bayesian optimization, in which we use an acquisition function, such as expected improvement or upper confidence bound, to rank the candidate models, rather than greedily picking the top K (see e.g., [222], [233]); adaptively varying the number of models K evaluated at each step (e.g., reducing it over time); the automatic exploration of speed-accuracy tradeoffs (cf., [54]), etc.

Chapter 3

Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation

This chapter aims at generalizing Neural Architecture Search to handle semantic image segmentation, or more generally, dense image prediction problems.

3.1 Introduction

Deep neural networks have been proved successful across a large variety of artificial intelligence tasks, including image recognition [89], [134], speech recognition [91], machine translation [242], [261] *etc*. While better optimizers [125] and better normalization techniques [111], [262] certainly played an important role, a lot of the progress comes from the design of neural network architectures. In computer vision, this holds true for both image classification [89], [96], [104], [134], [232], [244]–[246], [269] and dense image prediction [29], [64], [161], [188], [193], [213].

More recently, in the spirit of AutoML and democratizing AI, there has been significant interest in designing neural network architectures *automatically*, instead of relying heavily on expert experience and knowledge. Importantly, in the past year, Neural Architecture Search (NAS) has successfully identified architectures that exceed human-designed

Model	Auto Search				Task
	Cell	Network	Dataset	Days	
ResNet [89]	✗	✗	-	-	Cls
DenseNet [104]	✗	✗	-	-	Cls
DeepLabv3+ [32]	✗	✗	-	-	Seg
NASNet [298]	✓	✗	CIFAR-10	2000	Cls
AmoebaNet [208]	✓	✗	CIFAR-10	2000	Cls
PNASNet [154]	✓	✗	CIFAR-10	150	Cls
DARTS [157]	✓	✗	CIFAR-10	4	Cls
DPC [28]	✓	✗	Cityscapes	2600	Seg
Auto-DeepLab	✓	✓	Cityscapes	3	Seg

Table 3.1. Comparing our work against other CNN architectures with two-level hierarchy. The main differences include: (1) we directly search CNN architecture for semantic segmentation, (2) we search the network level architecture as well as the cell level one, and (3) our efficient search only requires 3 P100 GPU days.

architectures on large-scale image classification problems [154], [208], [298].

Image classification is a good starting point for NAS, because it is the most fundamental and well-studied high-level recognition task. In addition, there exists benchmark datasets (*e.g.*, CIFAR-10) with relatively small images, resulting in less computation and faster training. However, image classification should not be the end point for NAS, and the current success shows promise to extend into more demanding domains. In this chapter, we study Neural Architecture Search for semantic image segmentation, an important computer vision task that assigns a label like "person" or "bicycle" to each pixel in the input image.

Naively porting ideas from image classification would not suffice for semantic segmentation. In image classification, NAS typically applies transfer learning from low resolution images to high resolution images [298], whereas optimal architectures for semantic segmentation must inherently operate on high resolution imagery. This suggests the need for: (1) a more relaxed and general search space to capture the architectural variations brought

by the higher resolution, and (2) a more efficient architecture search technique as higher resolution requires heavier computation.

We notice that modern CNN designs [89], [104], [269] usually follow a two-level hierarchy, where the outer network level controls the spatial resolution changes, and the inner cell level governs the specific layer-wise computations. The vast majority of current works on NAS [154], [157], [202], [208], [298] follow this two-level hierarchical design, but only automatically search the inner cell level while hand-designing the outer network level. This limited search space becomes problematic for dense image prediction, which is sensitive to the spatial resolution changes. Therefore in our work, we propose a trellis-like network level search space that augments the commonly-used cell level search space first proposed in [298] to form a *hierarchical* architecture search space. Our goal is to jointly learn a good combination of repeatable cell structure and network structure specifically for semantic image segmentation.

In terms of the architecture search method, reinforcement learning [297], [298] and evolutionary algorithms [208], [210] tend to be computationally intensive even on the low resolution CIFAR-10 dataset, therefore probably not suitable for semantic image segmentation. We draw inspiration from the differentiable formulation of NAS [157], [227], and develop a continuous relaxation of the discrete architectures that exactly matches the hierarchical architecture search space. The hierarchical architecture search is conducted via stochastic gradient descent. When the search terminates, the best cell architecture is decoded greedily, and the best network architecture is decoded efficiently using the Viterbi algorithm. We directly search architecture on 321×321 image crops from Cityscapes [42]. The search is very efficient and only takes about 3 days on one P100 GPU.

We report experimental results on multiple semantic segmentation benchmarks, including Cityscapes [42], PASCAL VOC 2012 [60], and ADE20K [294]. Without ImageNet [214] pretraining, our best model significantly outperforms FRRN-B [204] by 8.6% and GridNet [67] by 10.9% on Cityscapes test set, and performs comparably with other ImageNet-

pretrained state-of-the-art models [19], [28], [32], [264], [292] when also exploiting the coarse annotations on Cityscapes. Notably, our best model (without pretraining) attains the same performance as DeepLabv3+ [32] (with pretraining) while being 2.23 times faster in Multi-Adds. Additionally, our light-weight model attains the performance only 1.2% lower than DeepLabv3+ [32], while requiring 76.7% fewer parameters and being 4.65 times faster in Multi-Adds. Finally, on PASCAL VOC 2012 and ADE20K, our best model outperforms several state-of-the-art models [146], [264], [266], [292], [294] while using strictly less data for pretraining.

To summarize, the contribution of this chapter is four-fold:

- Ours is one of the first attempts to extend NAS beyond image classification to dense image prediction.
- We propose a network level architecture search space that augments and complements the much-studied cell level one, and consider the more challenging joint search of network level and cell level architectures.
- We develop a differentiable, continuous formulation that conducts the two-level hierarchical architecture search efficiently in 3 GPU days.
- Without ImageNet pretraining, our model significantly outperforms FRRN-B and GridNet, and attains comparable performance with other ImageNet-pretrained state-of-the-art models on Cityscapes. On PASCAL VOC 2012 and ADE20K, our best model also outperforms several state-of-the-art models.

3.2 Related Work

Semantic Image Segmentation Convolutional neural networks [142] deployed in a fully convolutional manner (FCNs [161], [220]) have achieved remarkable performance on several semantic segmentation benchmarks. Within the state-of-the-art systems, there

are two essential components: multi-scale context module and neural network design. It has been known that context information is crucial for pixel labeling tasks. Therefore, PSPNet [292] performs spatial pyramid pooling [81], [87], [141] at several grid scales (including image-level pooling [159]), while DeepLab [30], [31] applies several parallel atrous convolution [29], [77], [94], [196], [220] with different rates. On the other hand, the improvement of neural network design has significantly driven the performance from AlexNet [134], VGG [232], Inception [111], [244], [246], ResNet [89] to more recent architectures, such as Wide ResNet [286], ResNeXt [269], DenseNet [104] and Xception [39]. In addition to adopting those networks as backbones for semantic segmentation, one could employ the encoder-decoder structures which efficiently captures the long-range context information while keeping the detailed object boundaries. Nevertheless, most of the models require initialization from the ImageNet [214] pretrained checkpoints except FRRN [204] and GridNet [67] for the task of semantic segmentation. Specifically, FRRN [204] employs a two-stream system, where full-resolution information is carried in one stream and context information in the other pooling stream. GridNet, building on top of a similar idea, contains multiple streams with different resolutions. In this work, we apply neural architecture search for network backbones specific for semantic segmentation. We further show state-of-the-art performance without ImageNet pretraining, and significantly outperforms FRRN [204] and GridNet [67] on Cityscapes [42].

Neural Architecture Search Method Neural Architecture Search aims at automatically designing neural network architectures, hence minimizing human hours and efforts. While some works [82], [115], [157], [297] search RNN cells for language tasks, more works search good CNN architectures for image classification.

Several papers used reinforcement learning (either policy gradients [21], [247], [297], [298] or Q-learning [12], [293]) to train a recurrent neural network that represents a policy to generate a sequence of symbols specifying the CNN architecture. An alternative to

RL is to use evolutionary algorithms (EA), that "evolves" architectures by mutating the best architectures found so far [156], [178], [208], [210], [268]. However, these RL and EA methods tend to require massive computation during the search, usually thousands of GPU days. PNAS [154] proposed a progressive search strategy that markedly reduced the search cost while maintaining the quality of the searched architecture. NAO [164] embedded architectures into a latent space and performed optimization before decoding. Additionally, several works [1], [157], [202] utilized architectural sharing among sampled models instead of training each of them individually, thereby further reduced the search cost. Our work follows the differentiable NAS formulation [157], [227] and extends it into the more general hierarchical setting.

Neural Architecture Search Space Earlier papers, *e.g.*, [210], [297], tried to directly construct the entire network. However, more recent papers [154], [157], [202], [208], [298] have shifted to searching the repeatable cell structure, while keeping the outer network level structure fixed by hand. First proposed in [298], this strategy is likely inspired by the two-level hierarchy commonly used in modern CNNs.

Our work still uses this cell level search space to keep consistent with previous works. Yet one of our contributions is to propose a new, general-purpose network level search space, since we wish to jointly search across this two-level hierarchy. Our network level search space shares a similar outlook as [217], but the important difference is that [217] kept the entire "fabrics" with no intention to alter the architecture, whereas we associate an explicit weight for each connection and focus on decoding a *single* discrete structure. In addition, [217] was evaluated on segmenting face images into 3 classes [116], whereas our models are evaluated on large-scale segmentation datasets such as Cityscapes [42], PASCAL VOC 2012 [60], and ADE20K [294].

The most similar work to ours is [28], which also studied NAS for semantic image segmentation. However, [28] focused on searching the much smaller Atrous Spatial

Pyramid Pooling (ASPP) module using random search, whereas we focus on searching the much more fundamental network backbone architecture using more advanced and more efficient search methods.

3.3 Architecture Search Space

This section describes our two-level hierarchical architecture search space. For the inner cell level (Section 3.3.1), we reuse the one adopted in [154], [157], [208], [298] to keep consistent with previous works. For the outer network level (Section 3.3.2), we propose a novel search space based on observation and summarization of many popular designs.

3.3.1 Cell Level Search Space

We define a *cell* to be a small fully convolutional module, typically repeated multiple times to form the entire neural network. More specifically, a cell is a directed acyclic graph consisting of B blocks.

Each *block* is a two-branch structure, mapping from 2 input tensors to 1 output tensor. Block i in cell l may be specified using a 5-tuple (I_1, I_2, O_1, O_2, C), where $I_1, I_2 \in \mathcal{I}_i^l$ are selections of input tensors, $O_1, O_2 \in \mathcal{O}$ are selections of layer types applied to the corresponding input tensor, and $C \in \mathcal{C}$ is the method used to combine the individual outputs of the two branches to form this block's output tensor, H_i^l. The cell's output tensor H^l is simply the concatenation of the blocks' output tensors $H_1^l, \ldots, H_B^l$ in this order.

The set of possible input tensors, $\mathcal{I}_i^l$, consists of the output of the previous cell H^{l-1}, the output of the previous-previous cell H^{l-2}, and previous blocks' output in the current cell $\{H_1^l, \ldots, H_i^l\}$. Therefore, as we add more blocks in the cell, the next block has more choices as potential source of input.

The set of possible layer types, $\mathcal{O}$, consists of the following 8 operators, all prevalent in modern CNNs:

- 3×3 depthwise-separable conv
- 5×5 depthwise-separable conv
- 3×3 atrous conv with rate 2
- 5×5 atrous conv with rate 2
- 3×3 average pooling
- 3×3 max pooling
- skip connection
- no connection (zero)

For the set of possible combination operators $\mathcal{C}$, we simply let element-wise addition to be the only choice.

3.3.2 Network Level Search Space

In the image classification NAS framework pioneered by [298], once a cell structure is found, the entire network is constructed using a pre-defined pattern. Therefore the network level was not part of the architecture search, hence its search space has never been proposed nor designed.

This pre-defined pattern is simple and straightforward: a number of "normal cells" (cells that keep the spatial resolution of the feature tensor) are separated equally by inserting "reduction cells" (cells that divide the spatial resolution by 2 and multiply the number of filters by 2). This keep-downsampling strategy is reasonable in the image classification case, but in dense image prediction it is also important to keep high spatial resolution, and as a result there are more network level variations [31], [188], [193].

Among the various network architectures for dense image prediction, we notice two principles that are consistent:

- The spatial resolution of the next layer is either twice as large, or twice as small, or remains the same.
- The smallest spatial resolution is downsampled by 32.

Following these common practices, we propose the following network level search space. The beginning of the network is a two-layer "stem" structure that each reduces the spatial

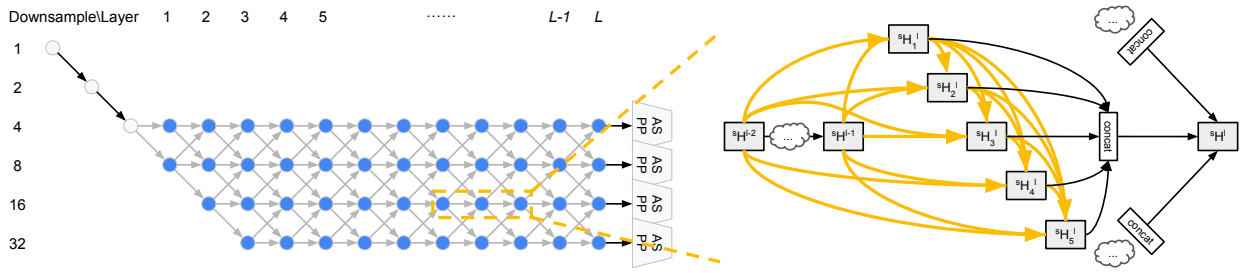


Figure 3.1. *Left:* Our network level search space with $L = 12$. Gray nodes represent the fixed "stem" layers, and a path along the blue nodes represents a candidate network level architecture. *Right:* During the search, each cell is a densely connected structure as described in Section 3.4.1.1. Every yellow arrow is associated with the set of values $\alpha_{j\to i}$. The three arrows after `concat` are associated with $\beta^l_{\frac{s}{2}\to s}, \beta^l_{s\to s}, \beta^l_{2s\to s}$ respectively, as described in Section 3.4.1.2. Best viewed in color.

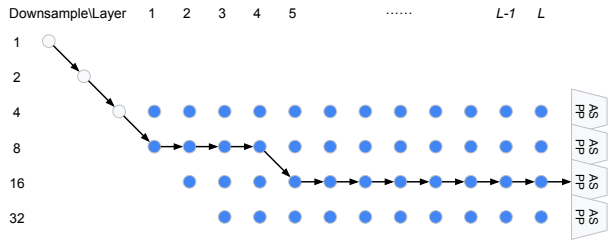
resolution by a factor of 2. After that, there are a total of L layers with unknown spatial resolutions, with the maximum being downsampled by 4 and the minimum being downsampled by 32. Since each layer may differ in spatial resolution by at most 2, the first layer after the stem could only be either downsampled by 4 or 8. We illustrate our network level search space in Figure 3.1. Our goal is then to find a good path in this L-layer trellis.

In Figure 3.2 we show that our search space is general enough to cover many popular designs. In the future, we have plans to relax this search space even further to include U-net architectures [147], [213], [228], where layer l may receive input from one more layer preceding l in addition to $l - 1$.

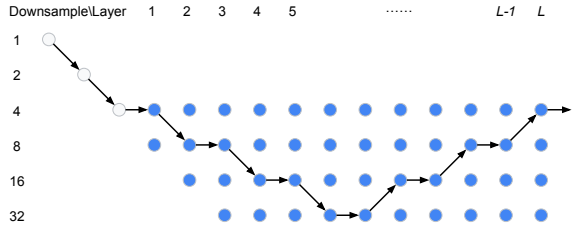
We reiterate that our work searches the network level architecture *in addition to* the cell level architecture. Therefore our search space is strictly more challenging and general-purpose than previous works.

3.4 Methods

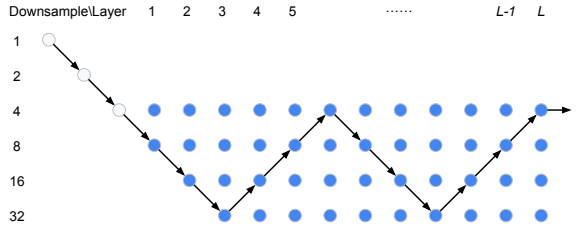
We begin by introducing a continuous relaxation of the (exponentially many) discrete architectures that exactly matches the hierarchical architecture search described above. We then discuss how to perform architecture search via optimization, and how to decode back



(a) Network level architecture used in DeepLabv3 [31].



(b) Network level architecture used in Conv-Deconv [193].



(c) Network level architecture used in Stacked Hourglass [188].

Figure 3.2. Our network level search space is general and includes various existing designs.

a discrete architecture after the search terminates.

3.4.1 Continuous Relaxation of Architectures

3.4.1.1 Cell Architecture

We reuse the continuous relaxation described in [157]. Every block's output tensor H_i^l is connected to all hidden states in $\mathcal{I}_i^l$:

$$H_i^l = \sum_{H_j^l \in \mathcal{I}_i^l} O_{j \to i}(H_j^l) \tag{3.1}$$

In addition, we approximate each $O_{j\to i}$ with its continuous relaxation $\bar{O}_{j\to i}$, defined as:

$$\bar{O}_{j\to i}(H_j^l) = \sum_{O^k \in \mathcal{O}} \alpha_{j\to i}^k O^k(H_j^l) \tag{3.2}$$

where

$$\sum_{k=1}^{|\mathcal{O}|} \alpha_{j\to i}^k = 1 \qquad \forall i, j \tag{3.3}$$

$$\alpha_{j\to i}^k \geq 0 \qquad \forall i, j, k \tag{3.4}$$

In other words, $\alpha_{j\to i}^k$ are normalized scalars associated with each operator $O^k \in \mathcal{O}$, easily implemented as softmax.

Recall from Section 3.3.1 that H^{l-1} and H^{l-2} are always included in $\mathcal{I}_i^l$, and that H^l is the concatenation of $H_1^l, \ldots, H_B^l$. Together with Equation (3.1) and Equation (3.2), the cell level update may be summarized as:

$$H^l = \text{Cell}(H^{l-1}, H^{l-2}; \alpha) \tag{3.5}$$

3.4.1.2 Network Architecture

Within a cell, all tensors are of the same spatial size, which enables the (weighted) sum in Equation (3.1) and Equation (3.2). However, as clearly illustrated in Figure 3.1, tensors may take different sizes in the network level. Therefore in order to set up the continuous relaxation, each layer l will have at most 4 hidden states $\{{}^4H^l, {}^8H^l, {}^{16}H^l, {}^{32}H^l\}$, with the upper left superscript indicating the spatial resolution.

We design the network level continuous relaxation to exactly match the search space described in Section 3.3.2. We associated a scalar with each gray arrow in Figure 3.1, and

the network level update is:

$$
\begin{aligned}
{}^{s}H^{l} = & \beta^{l}_{\frac{s}{2}\to s}\text{Cell}({}^{\frac{s}{2}}H^{l-1}, {}^{s}H^{l-2}; \alpha) \\
& + \beta^{l}_{s\to s}\text{Cell}({}^{s}H^{l-1}, {}^{s}H^{l-2}; \alpha) \\
& + \beta^{l}_{2s\to s}\text{Cell}({}^{2s}H^{l-1}, {}^{s}H^{l-2}; \alpha)
\end{aligned} \tag{3.6}
$$

where $s = 4, 8, 16, 32$ and $l = 1, 2, \ldots, L$. The scalars β are normalized such that

$$
\beta^{l}_{s\to\frac{s}{2}} + \beta^{l}_{s\to s} + \beta^{l}_{s\to 2s} = 1 \qquad \forall s, l \tag{3.7}
$$

$$
\beta^{l}_{s\to\frac{s}{2}} \geq 0 \quad \beta^{l}_{s\to s} \geq 0 \quad \beta^{l}_{s\to 2s} \geq 0 \qquad \forall s, l \tag{3.8}
$$

also implemented as softmax.

Equation (3.6) shows how the continuous relaxations of the two-level hierarchy are weaved together. In particular, β controls the outer network level, hence depends on the spatial size and layer index. Each scalar in β governs an entire set of α, yet α specifies the same architecture that depends on neither spatial size nor layer index.

As illustrated in Figure 3.1, Atrous Spatial Pyramid Pooling (ASPP) modules are attached to each spatial resolution at the L-th layer (atrous rates are adjusted accordingly). Their outputs are bilinear upsampled to the original resolution before summed to produce the prediction.

3.4.2 Optimization

The advantage of introducing this continuous relaxation is that the scalars controlling the connection strength between different hidden states are now part of the differentiable computation graph. Therefore they can be optimized efficiently using gradient descent. We adopt the first-order approximation in [157], and partition the training data into two disjoint sets *trainA* and *trainB*. The optimization alternates between:

1. Update network weights w by $\nabla_w \mathcal{L}_{trainA}(w, \alpha, \beta)$

2. Update architecture α, β by $\nabla_{\alpha,\beta}\mathcal{L}_{trainB}(w, \alpha, \beta)$

where the loss function $\mathcal{L}$ is the cross entropy calculated on the semantic segmentation mini-batch. The disjoint set partition is to prevent the architecture from overfitting the training data.

3.4.3 Decoding Discrete Architectures

Cell Architecture Following [157], we decode the discrete cell architecture by first retaining the 2 strongest predecessors for each block (with the strength from hidden state j to hidden state i being $\max_{k,O^k \neq zero} \alpha^k_{j\to i}$; recall from Section 3.3.1 that "zero" means "no connection"), and then choose the most likely operator by taking the argmax.

Network Architecture Equation (3.7) essentially states that the "outgoing probability" at each of the blue nodes in Figure 3.1 sums to 1. In fact, the β values can be interpreted as the "transition probability" between different "states" (spatial resolution) across different "time steps" (layer number). Quite intuitively, our goal is to find the path with the "maximum probability" from start to end. This path can be decoded efficiently using the classic Viterbi algorithm, as in our implementation.

3.5 Experimental Results

Herein, we report our architecture search implementation details as well as the search results. We then report semantic segmentation results on benchmark datasets with our best found architecture.

3.5.1 Architecture Search Implementation Details

We consider a total of $L = 12$ layers in the network, and $B = 5$ blocks in a cell. The network level search space has 2.9×10^4 unique paths, and the number of cell structures is

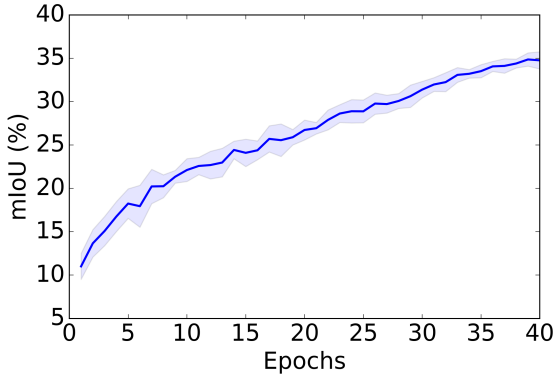


Figure 3.3. Validation accuracy during 40 epochs of architecture search optimization across 10 random trials.

5.6×10^{14}. So the size of the joint, hierarchical search space is in the order of 10^{19}.

We follow the common practice of doubling the number of filters when halving the height and width of feature tensor. Every blue node in Figure 3.1 with downsample rate s has $B \times F \times \frac{s}{4}$ output filters, where F is the filter multiplier controlling the model capacity. We set $F = 8$ during the architecture search. A stride 2 convolution is used for all $\frac{s}{2} \rightarrow s$ connections, both to reduce spatial size and double the number of filters. Bilinear upsampling followed by 1×1 convolution is used for all $2s \rightarrow s$ connections, both to increase spatial size and halve the number of filters.

The Atrous Spatial Pyramid Pooling module used in [31] has 5 branches: one 1×1 convolution, three 3×3 convolution with various atrous rates, and pooled image feature. During the search, we simplify ASPP to have 3 branches instead of 5 by only using one 3×3 convolution with atrous rate $\frac{96}{s}$. The number of filters produced by each ASPP branch is still $B \times F \times \frac{s}{4}$.

We conduct architecture search on the Cityscapes dataset [42] for semantic image segmentation. More specifically, we use 321×321 random image crops from half-resolution (512×1024) images in the *train_fine* set. We randomly select half of the images in *train_fine* as *trainA*, and the other half as *trainB* (see Section 3.4.2).

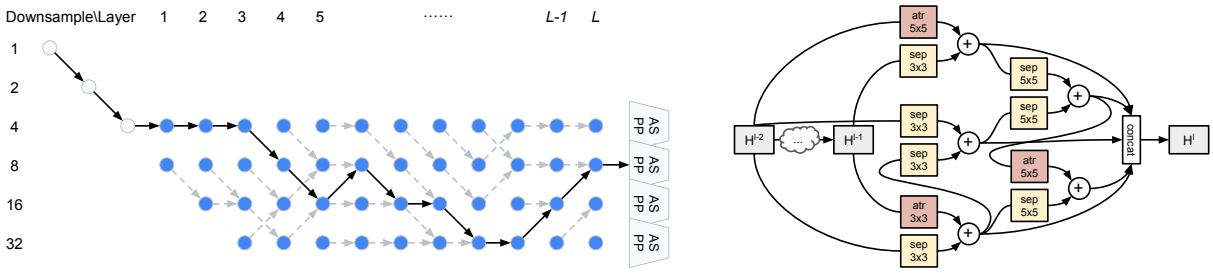


Figure 3.4. The Auto-DeepLab architecture found by our Hierarchical Neural Architecture Search on Cityscapes. Gray dashed arrows show the connection with maximum β at each node. **atr:** atrous convolution. **sep:** depthwise-separable convolution.

The architecture search optimization is conducted for a total of 40 epochs. The batch size is 2 due to GPU memory constraint. When learning network weights w, we use SGD optimizer with momentum 0.9, cosine learning rate that decays from 0.025 to 0.001, and weight decay 0.0003. The initial values of α, β before softmax are sampled from a standard Gaussian times 0.001. They are optimized using Adam optimizer [125] with learning rate 0.003 and weight decay 0.001. We empirically found that if α, β are optimized from the beginning when w are not well trained, the architecture tends to fall into bad local optima. Therefore we start optimizing α, β after 20 epochs. The entire architecture search optimization takes about 3 days on one P100 GPU. Figure 3.3 shows that the validation accuracy steadily improves throughout this process. We also tried searching for longer epochs (60, 80, 100), but did not observe benefit.

Figure 3.4 visualizes the best architecture found. In terms of network level architecture, higher resolution is preferred at both beginning (stays at downsample by 4 for longer) and end (ends at downsample by 8). We also show the strongest outgoing connection at each node using gray dashed arrows. We observe a general tendency to downsample in the first $3/4$ layers and upsample in the last $1/4$ layers. In terms of cell level architecture, the conjunction of atrous convolution and depthwise-separable convolution is often used, suggesting that the importance of context has been learned. Note that atrous convolution

Method	ImageNet	F	Multi-Adds	Params	mIOU (%)
Auto-DeepLab-S		20	333.25B	10.15M	79.74
Auto-DeepLab-M		32	460.93B	21.62M	80.04
Auto-DeepLab-L		48	695.03B	44.42M	80.33
FRRN-A [204]		-	-	17.76M	65.7
FRRN-B [204]		-	-	24.78M	-
DeepLabv3+ [32]	✓	-	1551.05B	43.48M	79.55

Table 3.2. Cityscapes validation set results with different Auto-DeepLab model variants. F: the filter multiplier controlling the model capacity. All our models are trained from *scratch* and with *single-scale* input during inference.

is rarely found to be useful in cells for image classification[1].

3.5.2 Semantic Segmentation Results

We evaluate the performance of our found best architecture (Figure 3.4) on Cityscapes [42], PASCAL VOC 2012 [60], and ADE20K [294] datasets.

We follow the same training protocol in [31], [32]. In brief, during training we adopt a polynomial learning rate schedule [159] with initial learning rate 0.05, and large crop size (*e.g.*, 769×769 on Cityscapes, and 513×513 on PASCAL VOC 2012 and resized ADE20K images). Batch normalization parameters [111] are fine-tuned during training. The models are trained from scratch with 1.5M iterations on Cityscapes, 1.5M iterations on PASCAL VOC 2012, and 4M iterations on ADE20K, respectively.

We adopt the simple encoder-decoder structure similar to DeepLabv3+ [32]. Specifically, our encoder consists of our found best network architecture augmented with the ASPP module [30], [31], and our decoder is the same as the one in DeepLabv3+ which recovers the boundary information by exploiting the low-level features that have downsample rate 4. Additionally, we redesign the "stem" structure with three 3×3 convolutions (with

[1] Among NASNet-{A, B, C}, PNASNet-{1, 2, 3, 4, 5}, AmoebaNet-{A, B, C}, ENAS, DARTS, atrous convolution was used only once in AmoebaNet-B reduction cell.

Method	itr-500K	itr-1M	itr-1.5M	**SDP**	mIOU (%)
Auto-DeepLab-S	✓				75.20
Auto-DeepLab-S		✓			77.09
Auto-DeepLab-S			✓		78.00
Auto-DeepLab-S			✓	✓	79.74

Table 3.3. Cityscapes validation set results. We experiment with the effect of adopting different training iterations (500K, 1M, and 1.5M iterations) and the Scheduled Drop Path method (**SDP**). All models are trained from scratch.

stride 2 in the first and third convolutions). The first two convolutions have 64 filters while the third convolution has 128 filters. This "stem" has been shown to be effective for segmentation in [255], [292].

3.5.2.1 Cityscapes

Cityscapes [42] contains high quality pixel-level annotations of 5000 images with size 1024×2048 (2975, 500, and 1525 for the training, validation, and test sets respectively) and about 20000 coarsely annotated training images. Following the evaluation protocol [42], 19 semantic labels are used for evaluation without considering the void label.

In Table 3.2, we report the Cityscapes validation set results. Similar to MobileNets [95], [215], we adjust the model capacity by changing the filter multiplier F. As shown in the table, higher model capacity leads to better performance at the cost of slower speed (indicated by larger Multi-Adds).

In Table 3.3, we show that increasing the training iterations from 500K to 1.5M iterations improves the performance by 2.8%, when employing our light-weight model variant, Auto-DeepLab-S. Additionally, adopting the Scheduled Drop Path [140], [298] further improves the performance by 1.74%, reaching 79.74% on Cityscapes validation set.

We then report the test set results in Table 3.4. Without any pretraining, our best model (Auto-DeepLab-L) significantly outperforms FRNN-B [204] by 8.6% and GridNet [67] by

Method	**ImageNet**	**Coarse**	mIOU (%)
FRRN-A [204]			63.0
GridNet [67]			69.5
FRRN-B [204]			71.8
Auto-DeepLab-S			79.9
Auto-DeepLab-L			80.4
Auto-DeepLab-S		✓	80.9
Auto-DeepLab-L		✓	82.1
ResNet-38 [264]	✓	✓	80.6
PSPNet [292]	✓	✓	81.2
Mapillary [19]	✓	✓	82.0
DeepLabv3+ [32]	✓	✓	82.1
DPC [28]	✓	✓	82.7
DRN_CRL_Coarse [296]	✓	✓	82.8

Table 3.4. Cityscapes test set results with *multi-scale* inputs during inference. **ImageNet:** Models pretrained on ImageNet. **Coarse:** Models exploit coarse annotations.

10.9%. With extra coarse annotations, our model Auto-DeepLab-L, without pretraining on ImageNet [214], achieves the test set performance of 82.1%, outperforming PSPNet [292] and Mapillary [19], and attains the same performance as DeepLabv3+ [32] while requiring 55.2% fewer Mutli-Adds computations. Notably, our light-weight model variant, Auto-DeepLab-S, attains 80.9% on the test set, comparable to PSPNet, while using merely 10.15M parameters and 333.25B Multi-Adds.

3.5.2.2 PASCAL VOC 2012

PASCAL VOC 2012 [60] contains 20 foreground object classes and one background class. We augment the original dataset with the extra annotations provided by [85], resulting in 10582 (*train_aug*) training images.

In Table 3.5, we report our validation set results. Our best model, Auto-DeepLab-L, with single scale inference significantly outperforms [73] by 20.36%. Additionally, for all our model variants, adopting multi-scale inference improves the performance by about 1%.

Method	**MS**	**COCO**	mIOU (%)
DropBlock [73]			53.4
Auto-DeepLab-S			71.68
Auto-DeepLab-S	✓		72.54
Auto-DeepLab-M			72.78
Auto-DeepLab-M	✓		73.69
Auto-DeepLab-L			73.76
Auto-DeepLab-L	✓		75.26
Auto-DeepLab-S		✓	78.31
Auto-DeepLab-S	✓	✓	80.27
Auto-DeepLab-M		✓	79.78
Auto-DeepLab-M	✓	✓	80.73
Auto-DeepLab-L		✓	80.75
Auto-DeepLab-L	✓	✓	82.04

Table 3.5. PASCAL VOC 2012 validation set results. We experiment with the effect of adopting *multi-scale* inference (**MS**) and COCO-pretrained checkpoints (**COCO**). Without any pretraining, our best model (Auto-DeepLab-L) outperforms DropBlock by 20.36%. All our models are not pretrained with ImageNet images.

Further pretraining our models on COCO [148] for 4M iterations improves the performance significantly.

Finally, we report the PASCAL VOC 2012 test set result with our COCO-pretrained model variants in Table 3.6. As shown in the table, our best model attains the performance of 85.6% on the test set, outperforming RefineNet [146] and PSPNet [292]. Our model is lagged behind the top-performing DeepLabv3+ [32] with Xception-65 as network backbone by 2.2%. We think that PASCAL VOC 2012 dataset is too small to train models from scratch and pretraining on ImageNet is still beneficial in this case.

3.5.2.3 ADE20K

ADE20K [294] has 150 semantic classes and high quality annotations of 20000 training images and 2000 validation images. In our experiments, the images are all resized so that

Method	**ImageNet**	**COCO**	mIOU (%)
Auto-DeepLab-S		✓	82.5
Auto-DeepLab-M		✓	84.1
Auto-DeepLab-L		✓	85.6
RefineNet [146]	✓	✓	84.2
ResNet-38 [264]	✓	✓	84.9
PSPNet [292]	✓	✓	85.4
DeepLabv3+ [32]	✓	✓	87.8
MSCI [145]	✓	✓	88.0

Table 3.6. PASCAL VOC 2012 test set results. Our Auto-DeepLab-L attains comparable performance with many state-of-the-art models which are pretrained on both **ImageNet** and **COCO** datasets. We refer readers to the official leader-board for other state-of-the-art models.

Method	**ImageNet**	mIOU (%)	Pixel-Acc (%)	Avg (%)
Auto-DeepLab-S		40.69	80.60	60.65
Auto-DeepLab-M		42.19	81.09	61.64
Auto-DeepLab-L		43.98	81.72	62.85
CascadeNet (VGG-16) [294]	✓	34.90	74.52	54.71
RefineNet (ResNet-152) [146]	✓	40.70	-	-
UPerNet (ResNet-101) [266] †	✓	42.66	81.01	61.84
PSPNet (ResNet-152) [292]	✓	43.51	81.38	62.45
PSPNet (ResNet-269) [292]	✓	44.94	81.69	63.32
DeepLabv3+ (Xception-65) [32] †	✓	45.65	82.52	64.09

Table 3.7. ADE20K validation set results. We employ *multi-scale* inputs during inference. †: Results are obtained from their up-to-date model zoo websites respectively. **ImageNet:** Models pretrained on ImageNet. Avg: Average of mIOU and Pixel-Accuracy.

the longer side is 513 during training.

In Table 3.7, we report our validation set results. Our models outperform some state-of-the-art models, including RefineNet [146], UPerNet [266], and PSPNet (ResNet-152) [292]; however, without any ImageNet [214] pretraining, our performance is lagged behind the latest work of [32].

3.6 Conclusion

In this chapter, we present one of the first attempts to extend Neural Architecture Search beyond image classification to dense image prediction problems. Instead of fixating on the cell level, we acknowledge the importance of spatial resolution changes, and embrace the architectural variations by incorporating the network level into the search space. We also develop a differentiable formulation that allows efficient (about $1000\times$ faster than DPC [28]) architecture search over our two-level hierarchical search space. The result of the search, Auto-DeepLab, is evaluated by training on benchmark semantic segmentation datasets from scratch. On Cityscapes, Auto-DeepLab significantly outperforms the previous state-of-the-art by 8.6%, and performs comparably with ImageNet-pretrained top models when exploiting the coarse annotations. On PASCAL VOC 2012 and ADE20K, Auto-DeepLab also outperforms several ImageNet-pretrained state-of-the-art models.

For future work, within the current framework, related applications such as object detection should be plausible; we could also try untying the cell architecture α across different layers (*c.f.* [247]) with little computation overhead. Beyond the current framework, a more general network level search space should be beneficial (*c.f.* Section 3.3.2).

Chapter 4

Are Labels Necessary for Neural Architecture Search?

This chapter extends Neural Architecture Search from the supervised regime to the unsupervised regime.

4.1 Introduction

Neural architecture search (NAS) has emerged as a research problem of searching for architectures that perform well on target data and tasks. A key mystery surrounding NAS is what factors contribute to the success of the search. Intuitively, using the target data and tasks during the search will result in the least domain gap, and this is indeed the strategy adopted in early NAS attempts [210], [297]. Later, researchers [298] started to utilize the transferability of architectures, which enabled the search to be performed on different data and labels (*e.g.*, CIFAR-10) than the target (*e.g.*, ImageNet). However, what has not changed is that *both the images and the (semantic) labels* provided in the dataset need to be used in order to search for an architecture. In other words, existing NAS approaches perform search in the *supervised* learning regime.

In this chapter, we take a step towards understanding what role supervision plays in the success of NAS. We ask the question: How indispensable are labels in neural architecture search? Is it possible to find high-quality architectures using images only? This corresponds

to the important yet underexplored *unsupervised setup* of neural architecture search, which we formalize in Section 4.3.

With the absence of labels, the quality of the architecture needs to be estimated in an unsupervised fashion during the search phase. In the present work, we conduct two sets of experiments using three unsupervised training methods [75], [194], [289] from the recent self-supervised learning literature.[1] These two sets of experiments approach the question from complementary perspectives. In *sample-based experiments*, we randomly sample 500 architectures from a search space, train and evaluate them using supervised *vs*. self-supervised objectives, and then examine the rank correlation (when sorting models by accuracy) between the two training methodologies. In *search-based experiments*, we take a well-established NAS algorithm, replace the supervised search objective with a self-supervised one, and examine the quality of the searched architecture on tasks such as ImageNet classification and Cityscapes semantic segmentation. Our findings include:

- The architecture rankings produced by supervised and self-supervised pretext tasks are *highly correlated*. This finding is consistent across two datasets, two search spaces, and three pretext tasks.
- The architectures searched without human annotations are *comparable in performance* to their supervised counterparts. This result is consistent across three pretext tasks, three pretext datasets, and two target tasks. There are even cases where unsupervised search outperforms supervised search.
- Existing NAS approaches typically use *labeled* images from a *smaller* dataset to learn transferable architectures. We present evidence that using *unlabeled* images from a *large* dataset may be a more promising approach.

We conclude that labels are not necessary for neural architecture search, and the

[1] Self-supervised learning is a form of unsupervised learning. The term "unsupervised learning" in general emphasizes "without *human*-annotated labels", while the term "self-supervised learning" emphasizes "producing labels from data".

deciding factor for architecture quality may hide within the image pixels.

4.2 Related Work

Neural Architecture Search. Research on the NAS problem involves designing the search space [279], [298] and the search algorithm [208], [297]. There are special focuses on reducing the overall time cost of the search process [154], [157], [202], or on extending to a larger variety of tasks [28], [74], [150], [234]. Existing works on NAS all use human-annotated labels during the search phase. Our work is orthogonal to existing NAS research, and advances in the existing NAS literature may also be applicable in our unsupervised setup.

Architecture Transferability. In early NAS attempts [210], [297], the search phase and the evaluation phase typically operate on the same dataset and task. Later, researchers realized that it is possible to relax this constraint. In these situations, the dataset and task used in the search phase are typically referred as the *proxy* to the target dataset and task, reflecting a notion of architecture transferability. [298] demonstrated that CIFAR-10 classification is a good proxy for ImageNet classification. [154] measured the rank correlation between these two tasks using a small number of architectures. [129] studied the transferability of 16 architectures (together with trained weights) between more *supervised* tasks. Part of our work studies architecture transferability at a larger scale, *across* supervised and unsupervised tasks.

Unsupervised Learning. There is a large literature on unsupervised learning, *e.g.*, [50], [55], [75], [86], [194], [195], [249], [258], [263], [289]. In the existing literature, methods are generally developed to learn the *weights* (parameters) of a fixed architecture without using labels, and these weights are evaluated by transferring to a target *supervised* task. In our study, we explore the possibility of using such methods to learn the *architecture* without

using labels, rather than the weights. Therefore, our subject of study is simultaneously the unsupervised generalization of NAS, and the architecture level generalization of unsupervised learning.

4.3 Unsupervised Neural Architecture Search

The goal of this chapter is to provide an answer to the question asked in the title: are labels necessary for neural architecture search? To formalize this question, in this section, we define a new setup called Unsupervised Neural Architecture Search (UnNAS). Note that UnNAS represents a general problem setup instead of any specific algorithm for solving this problem. We instantiate UnNAS with specific algorithms and experiments to explore the importance of labels in neural architecture search.

4.3.1 Search Phase

The traditional NAS problem includes a search phase: given a pre-defined search space, the search algorithm explores this space and estimates the performance (*e.g.* accuracy) of the architectures sampled from the space. The accuracy estimation can involve full or partial training of an architecture. Estimating the accuracy requires access to the labels of the dataset. So the traditional NAS problem is essentially a *supervised learning* problem.

We define UnNAS as the counterpart *unsupervised learning* problem. It follows the definition of the NAS problem, only except that there are no human-annotated labels provided for estimating the performance of the architectures. An algorithm for the UnNAS problem still explores a pre-defined search space,[2] but it requires other criteria to estimate how good a sampled architecture is.

[2] Admittedly, the search space design is typically heavily influenced by years of manual search, usually with supervision.

4.3.2 Evaluation Phase

Generally speaking, the goal of the NAS problem is to find an architecture. The weights of the found architecture are *not* necessarily the output of a NAS algorithm. Instead, the weights are optimized after the search phase in an evaluation phase of the NAS problem: it includes training the found architecture on a target dataset's training split, and validating the accuracy on the target dataset's validation split. We note that "*training* the architecture weights" is part of the NAS *evaluation* phase—the labels in the target dataset, both training and validation splits, play a role of evaluating an architecture.

Based on this context, we define *the evaluation phase of UnNAS* in the same way: training the weights of the architecture (found by an UnNAS algorithm) on a target dataset's training split, and validating the accuracy on the target dataset's validation split, both *using* the labels of the target dataset. We remark that using labels during the evaluation phase does *not* conflict with the definition of UnNAS: the search phase is unsupervised, while the evaluation phase requires labels to examine how good the architecture is.

4.3.3 Analogy to Unsupervised Learning

Our definition of UnNAS is analogous to unsupervised *weight* learning in existing literature [75], [194], [289]. While the unsupervised learning phase has no labels for training weights, the quality of the learned weights is evaluated by transferring them to a target task, supervised by labels. We emphasize that in the UnNAS setup, labels play an analogous role during evaluation. See Figure 4.1 for an illustration and elaboration on this analogy.

Similar to unsupervised weight learning, in principle, the search dataset should be different than the evaluation (target) dataset in UnNAS in order to more accurately reflect real application scenarios.

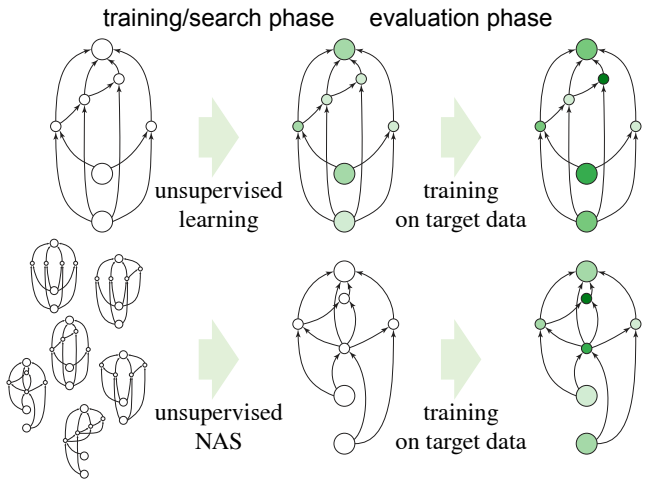


Figure 4.1. Unsupervised neural architecture search, or UnNAS, is a new problem setup that helps answer the question "are labels necessary for neural architecture search?" In traditional unsupervised learning (top panel), the *training phase* learns the weights of a fixed architecture; then the *evaluation phase* measures the quality of the weights by training a classifier (either by fine-tuning the weights or using them as a fixed feature extractor) using supervision from the target dataset. Analogously, in UnNAS (bottom panel), the *search phase* searches for an architecture without using labels; and the *evaluation phase* measures the quality of the architecture found by an UnNAS algorithm by training the architecture's weights using supervision from the target dataset.

4.4 Experiments Overview

As Section 4.3 describes, an architecture discovered in an *unsupervised* fashion will be evaluated by its performance in a *supervised* setting. Therefore, we are essentially looking for some type of architecture level *correlation* that can reach across the unsupervised *vs*. supervised boundary, so that the unsupervisedly discovered architecture could be reliably *transferred* to the supervised target task. We investigate whether several existing self-supervised pretext tasks (described in Section 4.4.1) can serve this purpose, through two sets of experiments of complementary nature: *sample-based* (Section 4.5) and *search-based* (Section 4.6). In *sample-based*, each network is trained and evaluated individually, but the downside is that we can only consider a small, random subset of the search space. In *search-based*, the focus is to find a top architecture from the entire search space, but the

downside is that the training dynamics during the search phase does not exactly match that of the evaluation phase.

4.4.1 Pretext Tasks

We explore three unsupervised training methods (typically referred to as *pretext tasks* in self-supervised learning literature): rotation prediction, colorization, and solving jigsaw puzzles. We briefly describe them for completeness.

- *Rotation prediction* [75] (`Rot`): the input image undergoes one of four preset rotations (0, 90, 180, and 270 degrees), and the pretext task is formulated as a 4-way classification problem that predicts the rotation.
- *Colorization* [289] (`Color`): the input is a grayscale image, and the pretext task is formulated as a pixel-wise classification problem with a set of pre-defined color classes (313 in [289]).
- *Solving jigsaw puzzles* [194] (`Jigsaw`): the input image is divided into patches and randomly shuffled. The pretext task is formulated as an image-wise classification problem that chooses from one out of K preset permutations.[3]

All three pretext tasks are image or pixel-wise classification problems. Therefore, we can compute the classification accuracy of the pretext task (on a validation set). Based on this pretext task accuracy, we can analyze its correlation with the supervised classification accuracy (also on a validation set), as in our sample-based experiments (Section 4.5). Also, since these pretext tasks all use cross entropy loss, it is also straightforward to use them as the training objective in standard NAS algorithms, as done in our search-based experiments (Section 4.6).

[3] On ImageNet, each image is divided into $3\times3=9$ patches and K is 1000 selected from $9!=362{,}880$ permutations [194]; on CIFAR-10 in which images are smaller, we use $2\times2=4$ patches and K is $4!=24$ permutations.

4.5 Sample-Based Experiments

4.5.1 Experimental Design

In *sample-based experiments*, we first randomly sample 500 architectures from a certain search space. We train each architecture from scratch on the pretext task and get its pretext task accuracy (*e.g.*, the 4-way classification accuracy in the rotation task), and also train the same architecture from scratch on the supervised classification task (*e.g.*, 1000-way classification on ImageNet).[4]

With these data collected, we perform two types of analysis. For the *rank correlation* analysis, we empirically study the statistical rank correlations between the pretext task accuracy and the supervised accuracy, by measuring the Spearman's Rank Correlation [236], denoted as ρ. For the *random experiment* analysis, we follow the setup proposed in [15] and recently adopted in [206]. Specifically, for each experiment size m, we sample m architectures from our pool of $n = 500$ architectures. For each pretext task, we select the architecture with the highest pretext task accuracy among the m. This process is repeated $\lceil n/m \rceil$ times, to compute the mean and error bands ($\pm$ 2 standard deviations) of the top-1 accuracy of these $\lceil n/m \rceil$ architectures *on the target dataset/task*.

These two studies provide complementary views. The *rank correlation* analysis aims to provide a global picture for *all* architectures sampled from a search space, while the *random experiment* analysis focuses on the *top* architectures in a random experiment of varying size.

We study two search spaces: the DARTS search space [157], and the NAS-Bench-101 search space [279]. The latter search space was built for benchmarking NAS algorithms, so we expect it to be less biased towards the search space for a certain algorithm. The experiments are conducted on two commonly used datasets: CIFAR-10 [133] and ImageNet [47].

[4] These architectures only have small, necessary differences when trained towards these different tasks (*i.e.*, having 4 neurons or 1000 neurons at the output layer).

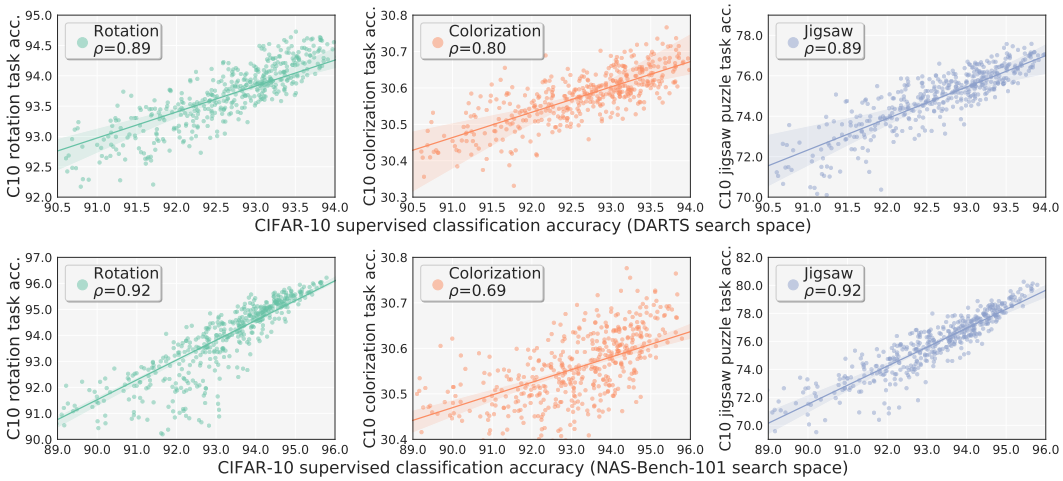


Figure 4.2. Correlation between supervised classification accuracy *vs.* pretext task accuracy on CIFAR-10 ("C10"). Top panel: DARTS search space. Bottom panel: NAS-Bench-101 search space. The straight lines are fit with robust linear regression [105] (same for Figure 4.3 and Figure 4.4).

4.5.2 Implementation Details

In the DARTS search space, regardless of the task, each network is trained for 5 epochs with width 32 and depth 22 on ImageNet; 100 epochs with width 16 and depth 20 on CIFAR-10. In the NAS-Bench-101 search space, regardless of the task, each network is trained for 10 epochs with width 128 and depth 12 on ImageNet; 100 epochs with width 128 and depth 9 on CIFAR-10. Please refer to [157], [279] for details on the overall network specifications in DARTS and NAS-Bench-101 search space, as well as the respective definitions of *width* and *depth*. The performance of each network on CIFAR-10 is the average of 3 independent runs to reduce variance.

We remark that the small-scale of CIFAR-10 dataset and the short training epochs on ImageNet are the compromises we make to allow for more diverse architectures (*i.e.* 500).

4.5.3 Results

High rank correlation between supervised accuracy and pretext accuracy on the same dataset. In Figure 4.2 and Figure 4.3, we show the scatter plots of the 500 architectures'

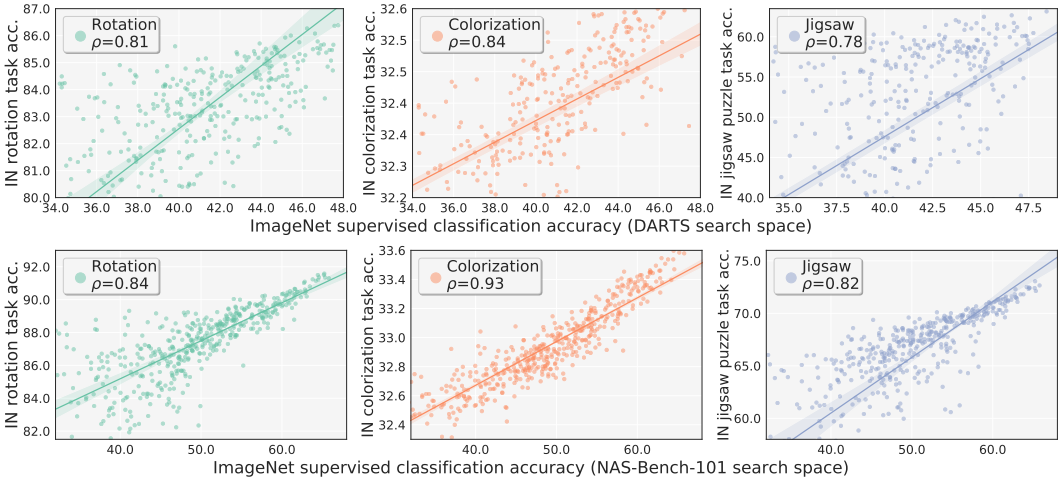


Figure 4.3. Correlation between supervised classification accuracy *vs.* pretext task accuracy on ImageNet ("IN"). Top panel: DARTS search space. Bottom panel: NAS-Bench-101 search space.

supervised classification accuracy (horizontal axis) and pretext task accuracy (vertical axis) on CIFAR-10 and ImageNet, respectively. We see that this rank correlation is typically higher than 0.8, regardless of the dataset, the search space, and the pretext task. This type of consistency and robustness indicates that this phenomenon is general, as opposed to dataset/search space specific.

The same experiment is performed on both the DARTS and the NAS-Bench-101 search spaces. The rank correlations on the NAS-Bench-101 search space are generally higher than those on the DARTS search space. A possible explanation is that the architectures in NAS-Bench-101 are more diverse, and consequently their accuracies have larger gaps, *i.e.*, are less affected by training noise.

Interestingly, we observe that among the three pretext tasks, colorization consistently has the lowest correlation on CIFAR-10, but the highest correlation on ImageNet. We suspect this is because the small images in CIFAR-10 make the learning of per-pixel colorization difficult, and consequently the performance after training is noisy.

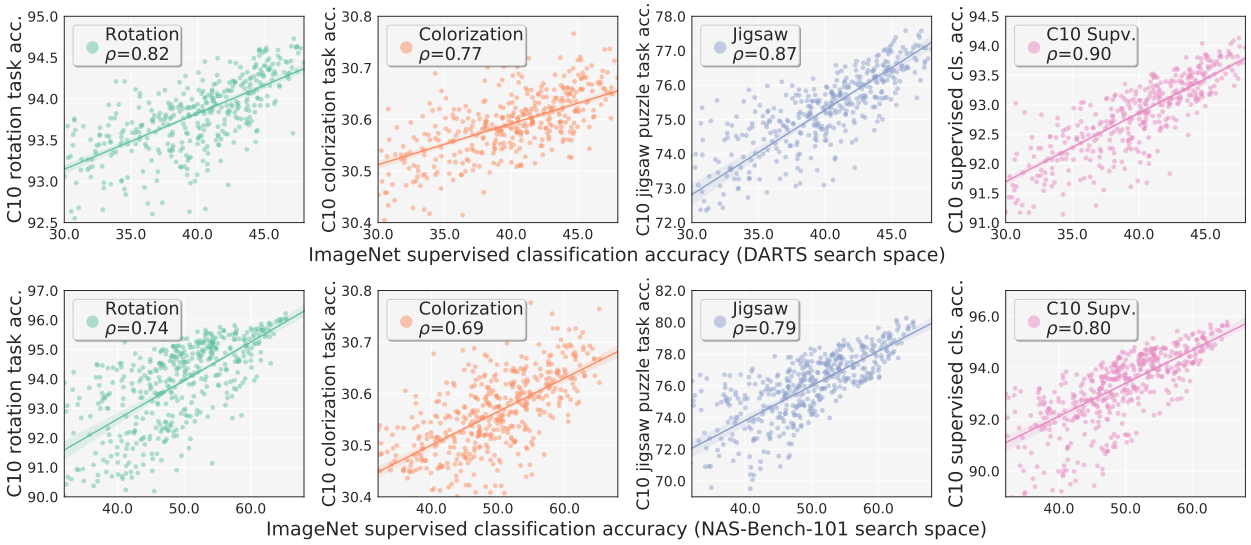


Figure 4.4. Correlation between ImageNet supervised classification accuracy *vs.* CIFAR-10 ("C10") pretext task accuracy. Rankings of architectures are highly correlated between supervised classification and three unsupervised tasks, as measured by Spearman's rank correlation (ρ). We also show rank correlation using CIFAR-10 supervised proxy in the rightmost panel. Top panel: DARTS search space. Bottom panel: NAS-Bench-101 search space.

High rank correlation between supervised accuracy and pretext accuracy across datasets. In Figure 4.4 we show the *across dataset* rank correlation analysis, where the pretext task accuracy is measured on CIFAR-10, but the supervised classification accuracy is measured on ImageNet.

On the DARTS search space (top panel), despite the image distribution shift brought by different datasets, for each of the three pretext tasks (left three plots), the correlation remains consistently high (~0.8). This shows that across the entire search space, the relative ranking of an architecture is likely to be similar, whether under the unsupervised pretext task accuracy or under the supervised target task accuracy. In the rightmost panel, we compare with a scenario where instead of using an unsupervised pretext task, we use a proxy task of CIFAR-10 supervised classification. The correlation is $\rho = 0.90$ in this case. As the CIFAR-10 supervised classification is a commonly used proxy task in existing NAS literature [298], it gives a reference on ρ's value.

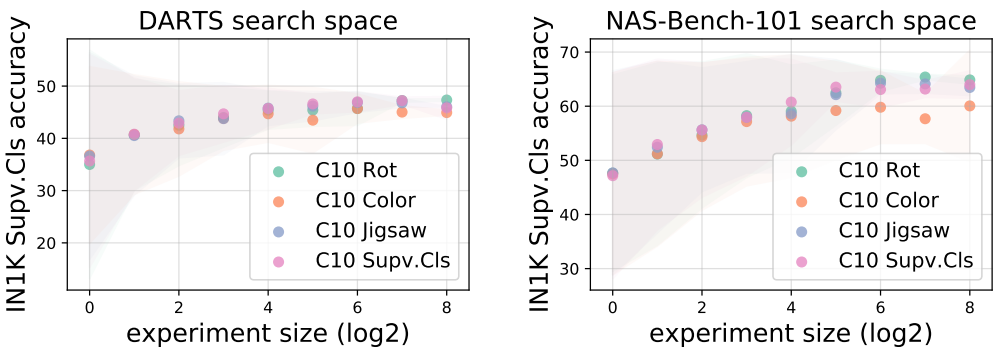


Figure 4.5. Random experiment efficiency curves. Left panel: DARTS search space. Right panel: NAS-Bench-101 search space. We show the range of ImageNet classification accuracies of top architectures identified by the three pretext tasks and the supervised task under various experiment sizes. See text for more details.

In the bottom panel we show more analysis of this kind by replacing the search space with NAS-Bench-101. Although the architectures are quite different, the observations are similar. In all cases, the self-supervised pretext task accuracy is highly correlated to the supervised classification accuracy.

Better pretext accuracy translates to better supervised accuracy. In addition to the rank correlation analysis, we also perform the random experiment analysis. Figure 4.5 shows the *random experiment efficiency* curve for DARTS and NAS-Bench-101 search spaces. Again, the pretext accuracies are obtained on CIFAR-10, and the target accuracies are from ImageNet. By design of this experiment, as the experiment size m increases, the *pretext* accuracies of the $\lceil n/m \rceil$ architectures should increase. Figure 4.5 shows that the *target* accuracies of these $\lceil n/m \rceil$ architectures also increase with m. In addition, at each experiment size, most unsupervised pretext objectives perform similarly compared to the commonly used supervised CIFAR-10 proxy. The overall trends are also comparable. This shows that the architecture rankings produced with and without labels are not only correlated across the entire search space, but also towards the top of the search space, which is closer to the goal of UnNAS.

4.6 Search-Based Experiments

4.6.1 Experimental Design

In *search-based experiments*, the idea is to run a well-established NAS algorithm, except that we make the minimal modification of replacing its supervised search objective with an unsupervised one. Following the UnNAS setup, we then examine (by training from scratch) how well these unsupervisedly discovered architectures perform on supervised target tasks. Since all other variables are controlled to be the same, search-based experiments can easily compare between the supervised and unsupervised counterparts of a NAS algorithm, which can help reveal the importance of labels.

The NAS algorithm we adopt is DARTS [157] (short for "differentiable architecture search") for its simplicity. DARTS formulates the activation tensor selection and operation selection as a categorical choice, implemented as a softmax function on a set of continuous parameters (named α). These parameters are trained in a similar fashion as the architecture weights, by backpropagation from a loss function. After this training, the softmax outputs are discretized and produce an architecture.

The self-supervised objectives that we consider are, still, those described in Section 4.4.1: rotation prediction (`Rot`), colorization (`Color`), and solving jigsaw puzzles (`Jigsaw`). For comparison, we also perform NAS search with supervised objectives, *e.g.* classification (`Supv.Cls`) or semantic segmentation (`Supv.Seg`). To help distinguish, we name the method NAS-`DARTS` if the search objective is supervised, and UnNAS-`DARTS` if the search objective is unsupervised.

4.6.2 Implementation Details

Search phase. We use three different datasets for architecture search: ImageNet-1K (IN1K) [47], ImageNet-22K (IN22K) [47], and Cityscapes [42]. IN1K is the standard ImageNet benchmark dataset with 1.2M images from 1K categories. IN22K is the full

ImageNet dataset that has ~14M images from 22K categories. Cityscapes is a dataset of street scenes that has drastically different image statistics. Note that UnNAS-`DARTS` will only access the images provided in the dataset, while NAS-`DARTS` will additionally access the (semantic) labels provided in the respective dataset. The search phase will operate only within the training split, without accessing the true validation or test split.

We report in Appendix B the hyper-parameters we used. One major difference between our experiments and DARTS [157] is that the images in the search datasets that we consider are much larger in size. We use 224×224 random crops for search on IN1K/IN22K, and 312×312 for search on Cityscapes following [150]. To enable DARTS training with large input images, we use 3 stride-2 convolution layers at the beginning of the network to reduce spatial resolution. This design, together with appropriately chosen number of search epochs (see Appendix B), allows UnNAS search to be efficient (~2 GPU days on IN1K/Cityscapes, ~10 GPU days on IN22K, regardless of task) despite running on larger images.

Evaluation phase. We use two distinct datasets and tasks for UnNAS evaluation: (1) ImageNet-1K (IN1K) for image classification. The performance metric is top-1 accuracy on the IN1K validation set. (2) Cityscapes for semantic segmentation. We use the `train_fine` set (2975 images) for training. The performance metric is mean Intersection over Union (mIoU) evaluated on the `val` set (500 images).

For IN1K evaluation, we fix depth to 14 and manually adjust the network width to have #FLOPs $\in$ [500,600]M. Models are trained for 250 epochs with an auxiliary loss weighted by 0.4, batch size 1024 across 8 GPUs, cosine learning rate schedule [162] with initial value 0.5, and 5 epochs of warmup. For Cityscapes evaluation, we fix depth to 12 and adjust the network width to have #Params $\in$ [9.5,10.5]M. We train the network for 2700 epochs, with batch size 64 across 8 GPUs, cosine learning rate schedule with initial value 0.1. For both ImageNet and Cityscapes evaluations, we report the mean and standard deviation of 3

independent trainings of the same architecture. More implementation details are described in Appendix B.

We note that under our definition of the UnNAS setup, the same dataset should not be used for both search and evaluation (because this scenario is unrealistic); We provide the IN1K→IN1K and Cityscapes→Cityscapes results purely as a reference. Those settings are analogous to the linear classifier probe for IN1K in conventional unsupervised learning research.

4.6.3 Results

In search-based experiments, the architectures are evaluated on both ImageNet classification, summarized in Table 4.1, and Cityscapes semantic segmentation, summarized in Table 4.2. We provide visualization of all NAS-`DARTS` and UnNAS-`DARTS` cell architectures in Appendix B.

UnNAS architectures perform competitively to supervised counterparts. We begin by comparing NAS-`DARTS` and UnNAS-`DARTS` when they are performed on the same search dataset. This would correspond to every four consecutive rows in Table 4.1 and Table 4.2, grouped together by horizontal lines.

As discussed earlier, strictly speaking the IN1K→IN1K experiment is not valid under our definition of UnNAS. For reference, we gray out these results in Table 4.1. NAS-`DARTS` on IN1K dataset has the highest performance among our experiments, achieving a top-1 accuracy of 76.3%. However, the UnNAS algorithm variants with `Rot`, `Color`, `Jigsaw` objectives all perform very well (achieving 75.8%, 75.7% and 75.9% top-1 accuracy, respectively), closely approaching the results obtained by the supervised counterpart. This suggests it might be desirable to perform architecture search on the target dataset directly, as also observed in other work [22].

Two valid UnNAS settings include IN22K→IN1K and Cityscapes→IN1K for archi-

method	search dataset & task	top-1 acc.	FLOPs (M)	params (M)
NAS-DARTS [157]	CIFAR-10 Supv.Cls	73.3	574	4.7
NAS-P-DARTS [35]	CIFAR-10 Supv.Cls	75.6	557	4.9
NAS-PC-DARTS [274]	CIFAR-10 Supv.Cls	74.9	586	5.3
NAS-PC-DARTS [274]	IN1K Supv.Cls	75.8	597	5.3
NAS-DARTS†	CIFAR-10 Supv.Cls	$74.9_{\pm 0.08}$	538	4.7
NAS-DARTS	IN1K Supv.Cls	$76.3_{\pm 0.06}$	590	5.3
UnNAS-DARTS	IN1K Rot	$75.8_{\pm 0.18}$	558	5.1
UnNAS-DARTS	IN1K Color	$75.7_{\pm 0.12}$	547	4.9
UnNAS-DARTS	IN1K Jigsaw	$75.9_{\pm 0.15}$	567	5.2
NAS-DARTS	IN22K Supv.Cls	$75.9_{\pm 0.09}$	585	5.2
UnNAS-DARTS	IN22K Rot	$75.7_{\pm 0.23}$	549	5.0
UnNAS-DARTS	IN22K Color	$75.9_{\pm 0.21}$	547	5.0
UnNAS-DARTS	IN22K Jigsaw	$75.9_{\pm 0.31}$	559	5.1
NAS-DARTS	Cityscapes Supv.Seg	$75.8_{\pm 0.13}$	566	5.1
UnNAS-DARTS	Cityscapes Rot	$75.9_{\pm 0.19}$	554	5.1
UnNAS-DARTS	Cityscapes Color	$75.2_{\pm 0.15}$	594	5.1
UnNAS-DARTS	Cityscapes Jigsaw	$75.5_{\pm 0.06}$	566	5.0

Table 4.1. ImageNet-1K classification results of the architectures searched by NAS and UnNAS algorithms. Rows in gray correspond to invalid UnNAS configurations where the search and evaluation datasets are the same. † is our training result of the DARTS architecture released in [157].

tecture search and evaluation. For IN22K→IN1K experiments, NAS and UnNAS results across the board are comparable. For Cityscapes→IN1K experiments, among the UnNAS architectures, Rot and Jigsaw perform well, once again achieving results comparable to the supervised search. However, there is a drop for UnNAS-DARTS search with Color objective (with a 75.2% top-1 accuracy). We hypothesize that this might be owing to the fact that the color distribution in Cityscapes images is not as diverse: the majority of the pixels are from *road* and *ground* categories of gray colors.

In general, the variances are higher for Cityscapes semantic segmentation (Table 4.2), but overall UnNAS-DARTS architectures still perform competitively to NAS-DARTS architectures, measured by semantic segmentation mIoU. For the Cityscapes→Cityscapes experiment, we observe that searching with segmentation objective directly leads to in-

method	search dataset & task	mIoU	FLOPs (B)	params (M)
NAS-DARTS†	CIFAR-10 Supv.Cls	$72.6_{\pm 0.55}$	121	9.6
NAS-DARTS	IN1K Supv.Cls	$73.6_{\pm 0.31}$	127	10.2
UnNAS-DARTS	IN1K Rot	$73.6_{\pm 0.29}$	129	10.4
UnNAS-DARTS	IN1K Color	$72.2_{\pm 0.56}$	122	9.7
UnNAS-DARTS	IN1K Jigsaw	$73.1_{\pm 0.17}$	129	10.4
NAS-DARTS	IN22K Supv.Cls	$72.4_{\pm 0.29}$	126	10.1
UnNAS-DARTS	IN22K Rot	$72.9_{\pm 0.23}$	128	10.3
UnNAS-DARTS	IN22K Color	$73.6_{\pm 0.41}$	128	10.3
UnNAS-DARTS	IN22K Jigsaw	$73.1_{\pm 0.59}$	129	10.4
NAS-DARTS	Cityscapes Supv.Seg	$72.4_{\pm 0.15}$	128	10.3
UnNAS-DARTS	Cityscapes Rot	$73.0_{\pm 0.25}$	128	10.3
UnNAS-DARTS	Cityscapes Color	$72.5_{\pm 0.31}$	122	9.5
UnNAS-DARTS	**Cityscapes Jigsaw**	$\mathbf{74.1}_{\pm 0.39}$	**128**	**10.2**

Table 4.2. Cityscapes semantic segmentation results of the architectures searched by NAS and UnNAS algorithms. These are trained from scratch: there is no fine-tuning from ImageNet checkpoint. Rows in gray correspond to an illegitimate setup where the search dataset is the same as the evaluation dataset. † is our training result of the DARTS architecture released in [157].

ferior result (mean 72.4% mIoU), compared to the architectures searched for ImageNet classification tasks. This is different from what has been observed in Table 4.1. However, under this setting, our UnNAS algorithm, in particular the one with the Jigsaw objective shows very promising results (mean 74.1% mIoU). In fact, when the search dataset is IN22K or Cityscapes, all UnNAS-DARTS architectures perform *better* than the NAS-DARTS architecture. This is the opposite of what was observed in IN1K→IN1K. Results for Cityscapes→Cityscapes are grayed out for the same reason as before (invalid under UnNAS definition).

NAS and UnNAS results are robust across a large variety of datasets and tasks. The three search datasets that we consider are of different nature. For example, IN22K is 10 times larger than IN1K, and Cityscapes images have a markedly different distribution than those in ImageNet. In our experiments, NAS-DARTS/UnNAS-DARTS architectures searched on IN22K do not significantly outperform those searched on IN1K, meaning that

they do not seem to be able to enjoy the benefit of having more abundant images. This reveals new opportunities in designing better algorithms to exploit bigger datasets for neural architecture search. For Cityscapes→IN1K experiments, it is interesting to see that after switching to a dataset with markedly distinct search images (urban street scenes), we are still able to observe decent performance. The same goes for the reverse direction IN1K/IN22K→Cityscapes, which implies that the search does not severely overfit to the images from the dataset.

In addition to this robustness to the *search dataset* distribution, NAS and UnNAS also exhibit robustness to *target dataset and task*. Classification on ImageNet and segmentation on Cityscapes are different in many ways, but among different combinations of search dataset and task (whether supervised or unsupervised), we do not observe a case where the same architecture performs well on one but poorly on the other.

UnNAS outperforms previous methods. Finally, we compare our UnNAS-DARTS results against existing works. We first note that we are able to achieve a better baseline number with the NAS-DARTS architecture (searched with the CIFAR-10 proxy) compared to what was reported in [157]. This is mainly due to better hyper-parameter tuning we adopt from [35] for the *evaluation phase* model training. This baseline sets up a fair ground for all the UnNAS experiments; we use the same evaluation phase hyper-parameters across different settings.

On ImageNet, our UnNAS-DARTS architectures can comfortably outperform this baseline by up to 1% classification accuracy. In fact, the extremely competitive UnNAS-DARTS results also outperform the previous best result (75.8%) on this search space, achieved with a more sophisticated NAS algorithm [274].

On Cityscapes, there have not been many works that use DARTS variants as the backbone. The closest is Auto-DeepLab [150], but we use a lighter architecture (in that we do not have the Decoder) and shorter training iterations, so the results are not directly

comparable. Nonetheless, according to our evaluation, the UnNAS architectures perform favorably against the DARTS architecture released in [157] (discovered with the CIFAR-10 proxy). The best UnNAS-`DARTS` variant (Cityscapes `Jigsaw`) achieves 74.1% mIoU, which outperforms this baseline by 1.5% on average. Overall, our experiments demonstrate that exploring neural architecture search with unsupervised/self-supervised objectives to improve target task performance might be a fruitful direction.

Outperforming previous methods was far from the original goal of our study. Nonetheless, the promising results of UnNAS suggest that in addition to developing new *algorithms* and finding new *tasks*, the role of *data* (in our case, more/larger images) and *paradigm* (in our case, no human annotations) is also worth attention in future work on neural architecture search.

4.7 Discussion

In this chapter, we challenge the common practice in neural architecture search and ask the question: do we really need labels to successfully perform NAS? We approach this question with two sets of experiments: sample-based and search-based. In sample-based experiments, by randomly sampling a large number of architectures, we discover the phenomenon that the architecture rankings produced with and without labels are highly correlated. In search-based experiments, by making minimal modifications to a well-established NAS algorithm, DARTS,[5] we show that the architectures learned without accessing labels perform competitively, not only relative to their supervised counterpart, but also in terms of absolute performance. In both experiments, the observations are consistent and robust across various datasets, tasks, and/or search spaces. Overall, the findings in this chapter indicate that labels are *not* necessary for neural architecture search.

[5] We are aware of the limitations of this algorithm, and do not imply that the search algorithm is a solved problem. In fact, this is part of the motivation for the sample-based experiments: to complement DARTS with discrete, individual architectures.

How to learn and transfer useful representations to subsequent tasks in an unsupervised fashion has been a research topic of extensive interest, but the discovery of neural network architectures has been driven solely by supervised tasks. As a result, current NAS products or AutoML APIs typically have the strict prerequisite for users to "*put together a training dataset of labeled images*" [79]. An immediate implication of our study is that the job of the user could potentially be made easier by dropping the labeling effort. In this sense, UnNAS could be especially beneficial to the many applications where data constantly comes in at large volume but labeling is costly.

At the same time, we should still ask: if not labels, then what factors are needed to reveal a good architecture? A meaningful unsupervised task seems to be important, though the several pretext tasks considered in this chapter do not exhibit significant difference in either of the two experiments. In the future we plan to investigate even more and even simpler unsupervised tasks. Another possibility is that the architecture quality is mainly decided by the image statistics, and since the datasets that we consider are all natural images, the correlations are high and the results are comparable. This hypothesis would also suggest an interesting, alternative direction: that instead of performing NAS again and again for every specific labeled task, it may be more sensible to perform NAS once on large amounts of unlabeled images that capture the image distribution.

Part II

Diagnosing Language-Level Understanding for 2D Images

Chapter 5

Attention Correctness in Neural Image Captioning

This chapter quantitatively examines the attention mechanism in neural image captioning.

5.1 Introduction

Recently, attention based deep models have been proved effective at handling a variety of AI problems such as machine translation [10], object detection [9], [182], visual question answering [27], [271], and image captioning [272]. Inspired by human attention mechanisms, these deep models learn dynamic weightings of the input vectors, which allow for more flexibility and expressive power.

In this work we focus on attention models for image captioning. The state-of-the-art image captioning models [53], [120], [127], [171], [253] adopt Convolutional Neural Networks (CNNs) to extract image features and Recurrent Neural Networks (RNNs) to decode these features into a sentence description. Within this encoder-decoder framework [38], the models proposed by [272] apply an attention mechanism, i.e. attending to different areas of the image when generating words one by one.

Although impressive visualization results of the attention maps for image captioning are shown in [272], the authors do not provide *quantitative evaluations* of the attention

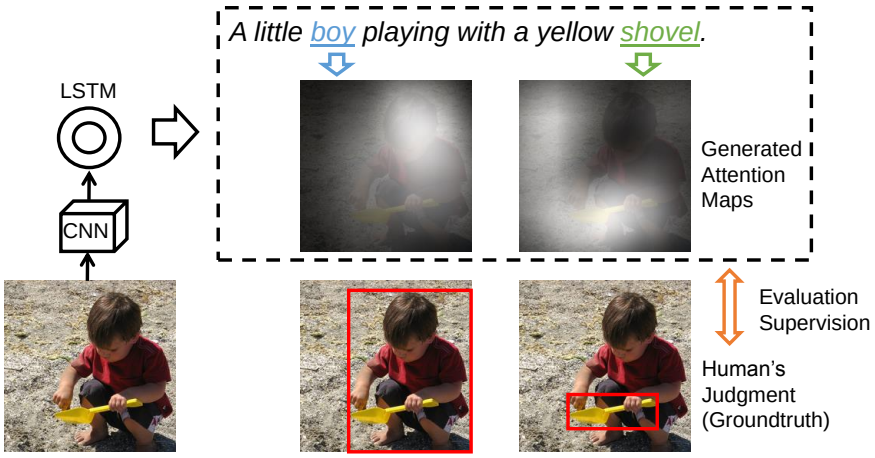


Figure 5.1. Image captioning models [272] can attend to different areas of the image when generating the words. However, these generated attention maps may not correspond to the region that the words or phrases describe in the image (e.g. "shovel"). We evaluate such phenomenon quantitatively by defining attention correctness, and alleviate this inconsistency by introducing explicit supervision. In addition, we show positive correlation between attention correctness and caption quality.

maps generated by their models. Since deep network attention can be viewed as a form of alignment from language space to image space, we argue that these attention maps in fact carry important information in understanding (and potentially improving) deep networks. Therefore in this chapter, we study the following two questions:

- How often and to what extent are the attention maps consistent with human perception/annotation?
- Will more human-like attention maps result in better captioning performance?

Towards these goals, we propose a novel quantitative metric to evaluate the "correctness" of attention maps. We define "correctness" as the consistency between the attention maps generated by the model and the corresponding region that the words/phrases describe in the image. More specifically, we use the alignment annotations between image regions and noun phrase caption entities provided in the Flickr30k Entities dataset [203] as our ground truth maps. Using this metric, we show that the attention model of [272]

performs better than the uniform attention baseline, but still has room for improvement in terms of attention consistency with human annotations.

Based on this observation, we propose a model with explicit supervision of the attention maps. The model can be used not only when detailed ground truth attention maps are given (e.g. the Flickr30k Entities dataset [203]) but also when only the semantic labelings of image regions (which is a much cheaper type of annotations) are available (e.g. MS COCO dataset [148]). Our experiments show that in both scenarios, our models perform consistently and significantly better than the implicit attention counterpart in terms of both attention maps accuracy and the quality of the final generated captions. To the best of our knowledge, this is the first work that quantitatively measures the quality of visual attention in deep models and shows significant improvement by adding supervision to the attention module.

5.2 Related Work

Image Captioning Models There has been growing interest in the field of image captioning, with lots of work demonstrating impressive results [36], [53], [63], [120], [127], [171], [253], [272]. However, it is uncertain to what extent the captioning models truly understand and recognize the objects in the image while generating the captions. [272] proposed an attention model and qualitatively showed that the model can attend to specific regions of the image by visualizing the attention maps of a few images. Our work takes a step further by quantitatively measuring the quality of the attention maps. The role of the attention maps also relates to referring expressions [102], [170], where the goal is predicting the part of the image that is relevant to the expression.

Deep Attention Models In machine translation, [10] introduced an extra softmax layer in the RNN/LSTM structure that generates weights of the individual words of the sentence to be translated. The quality of the attention/alignment was qualitatively visualized

in [10] and quantitatively evaluated in [166] using the alignment error rate. In image captioning, [272] used convolutional image features with spatial information as input, allowing attention on 2D space. [280] targeted attention on a set of concepts extracted from the image to generate image captions. In visual question answering, [27], [226], [271], [295] proposed several models which attend to image regions or questions when generating an answer. But none of these models quantitatively evaluates the quality of the attention maps or imposes supervision on the attention. Concurrently, [46] analyzed the consistency between human and deep network attention in visual question answering. Our goal differs in that we are interested in how attention changes with the progression of the description.

Image Description Datasets For image captioning, Flickr8k [93], Flickr30k [281], and MS COCO [148] are the most commonly used benchmark datasets. [203] developed the original caption annotations in Flickr30k by providing the region to phrase correspondences. Specifically, annotators were first asked to identify the noun phrases in the captions, and then mark the corresponding regions with bounding boxes. In this work we use this dataset as ground truth to evaluate the quality of the generated attention maps, as well as to train our strongly supervised attention model. Our model can also utilize the instance segmentation annotations in MS COCO to train our weakly supervised version.

5.3 Deep Attention Models for Image Captioning

In this section, we first discuss the attention model that learns the attention weights implicitly [272], and then introduce our explicit supervised attention model.

5.3.1 Implicit Attention Model

The implicit attention model [272] consists of three parts: the encoder which encodes the visual information (i.e. a visual feature extractor), the decoder which decodes the information into words, and the attention module which performs spatial attention.

The visual feature extractor produces L vectors that correspond to different spatial locations of the image: $a = \{\mathbf{a}_1, \ldots, \mathbf{a}_L\},\ \mathbf{a}_i \in \mathbb{R}^D$. Given the visual features, the goal of the decoder is to generate a caption y of length C: $y = \{y_1, \ldots, y_C\}$. We use $\mathbf{y}_t \in \mathbb{R}^K$ to represent the one-hot encoding of y_t, where K is the dictionary size.

In [272], an LSTM network [92] is used as the decoder:

$$\mathbf{i}_t = \sigma(W_i E\mathbf{y}_{t-1} + U_i\mathbf{h}_{t-1} + Z_i\mathbf{z}_t + \mathbf{b}_i) \tag{5.1}$$

$$\mathbf{f}_t = \sigma(W_f E\mathbf{y}_{t-1} + U_f\mathbf{h}_{t-1} + Z_f\mathbf{z}_t + \mathbf{b}_f) \tag{5.2}$$

$$\mathbf{c}_t = \mathbf{f}_t\mathbf{c}_{t-1} + \mathbf{i}_t\tanh(W_c E\mathbf{y}_{t-1} + U_c\mathbf{h}_{t-1} + Z_c\mathbf{z}_t + \mathbf{b}_c) \tag{5.3}$$

$$\mathbf{o}_t = \sigma(W_o E\mathbf{y}_{t-1} + U_o\mathbf{h}_{t-1} + Z_o\mathbf{z}_t + \mathbf{b}_o) \tag{5.4}$$

$$\mathbf{h}_t = \mathbf{o}_t\tanh(\mathbf{c}_t) \tag{5.5}$$

where $\mathbf{i}_t, \mathbf{f}_t, \mathbf{c}_t, \mathbf{o}_t, \mathbf{h}_t$ are input gate, forget gate, memory, output gate, and hidden state of the LSTM respectively. $W, U, Z, \mathbf{b}$ are weight matrices and biases. $E \in \mathbb{R}^{m\times K}$ is an embedding matrix, and σ is the sigmoid function. The context vector $\mathbf{z}_t = \sum_{i=1}^{L} \alpha_{ti}\mathbf{a}_i$ is a dynamic vector that represents the relevant part of image feature at time step t, where α_{ti} is a scalar weighting of visual vector $\mathbf{a}_i$ at time step t, defined as follows:

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{k=1}^{L}\exp(e_{tk})} \qquad e_{ti} = f_{attn}(\mathbf{a_i}, \mathbf{h}_{t-1}) \tag{5.6}$$

$f_{attn}(\mathbf{a_i}, \mathbf{h}_{t-1})$ is a function that determines the amount of attention allocated to image feature $\mathbf{a_i}$, conditioned on the LSTM hidden state $\mathbf{h}_{t-1}$. In [272], this function is implemented as a multilayer perceptron. Note that by construction $\sum_{i=1}^{L}\alpha_{ti} = 1$.

The output word probability is determined by the image $\mathbf{z}_t$, the previous word y_{t-1}, and the hidden state $\mathbf{h}_t$:

$$p(y_t|a, y_{t-1}) \propto \exp(G_o(E\mathbf{y}_{t-1} + G_h\mathbf{h}_t + G_z\mathbf{z}_t)) \tag{5.7}$$

where G are learned parameters. The loss function, ignoring the regularization terms, is

the negative log probability of the ground truth words $w = \{w_1, \ldots, w_C\}$:

$$L_{t,cap} = -\log p(w_t|a, y_{t-1}) \tag{5.8}$$

5.3.2 Supervised Attention Model

In this work we are interested in the attention map generated by the model $\boldsymbol{\alpha}_t = \{\alpha_{ti}\}_{i=1,\ldots,L}$. One limitation of the model in [272] is that even if we have some prior knowledge about the attention map, it will not be able to take advantage of this information to learn a better attention function $f_{attn}(\mathbf{a_i}, \mathbf{h}_{t-1})$. We tackle this problem by introducing explicit supervision.

Concretely, we first consider the case when the ground truth attention map $\boldsymbol{\beta}_t = \{\beta_{ti}\}_{i=1,\ldots,L}$ is provided for the ground truth word w_t, with $\sum_{i=1}^{L} \beta_{ti} = 1$. Since $\sum_{i=1}^{L} \beta_{ti} = \sum_{i=1}^{L} \alpha_{ti} = 1$, they can be considered as two probability distributions of attention and it is natural to use the cross entropy loss. For the words that do not have an alignment with an image region (e.g. "a", "is"), we simply set $L_{t,attn}$ to be 0:

$$L_{t,attn} = \begin{cases} -\sum_{i=1}^{L} \beta_{ti} \log \alpha_{ti} & \text{if } \boldsymbol{\beta}_t \text{ exists for } w_t \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

The total loss is the weighted sum of the two loss terms: $L = \sum_{t=1}^{C} L_{t,cap} + \lambda \sum_{t=1}^{C} L_{t,attn}$.

We then discuss two ways of constructing the ground truth attention map $\boldsymbol{\beta}_t$, depending on the types of annotations.

5.3.2.1 Strong Supervision with Alignment Annotation

In the simplest case, we have direct annotation that links the ground truth word w_t to a region R_t (in the form of bounding boxes or segmentation masks) in the image (e.g. Flickr30k Entities). We encourage the model to "attend to" R_t by constructing

$\hat{\boldsymbol{\beta}}_t = \{\hat{\beta}_{t\hat{i}}\}_{\hat{i}=1,..,\hat{L}}$ where:

$$\hat{\beta}_{t\hat{i}} = \begin{cases} 1 & \hat{i} \in R_t \\ 0 & \text{otherwise} \end{cases} \tag{5.10}$$

Note that the resolution of the region R (e.g. 224×224) and the attention map $\boldsymbol{\alpha}, \boldsymbol{\beta}$ (e.g. 14×14) may be different, so $\hat{L}$ could be different from L. Therefore we need to resize $\hat{\boldsymbol{\beta}}_t$ to the same resolution as $\boldsymbol{\alpha}_t$ and normalize it to get $\boldsymbol{\beta}_t$.

5.3.2.2 Weak Supervision with Semantic Labeling

Ground truth alignment is expensive to collect and annotate. A much more general and cheaper annotation is to use bounding boxes or segmentation masks with object class labels (e.g. MS COCO). In this case, we are provided with a set of regions R_j in the image with associated object classes c_j, $j = 1, \ldots, M$ where M is the number of object bounding boxes or segmentation masks in the image. Although not ideal, these annotations contain important information to guide the attention of the model. For instance, for the caption "a boy is playing with a dog", the model should attend to the region of a person when generating the word "boy", and attend to the region of a dog when generating the word "dog". This suggests that we can approximate image-to-language (region $\rightarrow$ word) consistency by language-to-language (object class $\rightarrow$ word) similarity.

Following this intuition, we set the likelihood that a word w_t and a region R_j are aligned by the similarity of w_t and c_j in the word embedding space:

$$\hat{\beta}_{t\hat{i}} = \begin{cases} \text{sim}(\tilde{E}(w_t), \tilde{E}(c_j)) & \hat{i} \in R_j \\ 0 & \text{otherwise} \end{cases} \tag{5.11}$$

where $\tilde{E}(w_t)$ and $\tilde{E}(c_j)$ denote the embeddings of the word w_t and c_j respectively. $\tilde{E}$ can be the embedding E learned by the model or any off-the-shelf word embedding (e.g. pre-trained word2vec). We then resize and normalize $\hat{\boldsymbol{\beta}}_t$ in the same way as the strong supervision scenario.

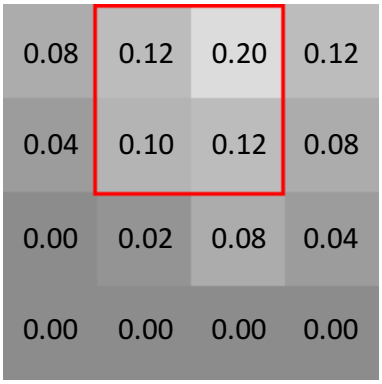


Figure 5.2. Attention correctness is the sum of the weights within ground truth region (red bounding box), in this illustration 0.12 + 0.20 + 0.10 + 0.12 = 0.54.

5.4 Attention Correctness: Evaluation Metric

At each time step in the implicit attention model, the LSTM not only predicts the next word y_t but also generates an attention map $\boldsymbol{\alpha_t} \in \mathbb{R}^L$ across all locations. However, the attention module is merely an intermediate step, while the error is only backpropagated from the word-likelihood loss in Equation 5.8. This opens the question of whether this implicitly-learned attention module is indeed effective.

Therefore in this section we introduce the concept of *attention correctness*, an evaluation metric that quantitatively analyzes the quality of the attention maps generated by the attention-based model.

5.4.1 Definition

For a word y_t with generated attention map $\boldsymbol{\alpha}_t$, let R_t be the ground truth attention region, then we define the word attention correctness by

$$AC(y_t) = \sum_{\hat{i} \in R_t} \hat{\alpha}_{t\hat{i}} \tag{5.12}$$

which is a score between 0 and 1. Intuitively, this value captures the sum of the attention score that falls within human annotation (see Figure 5.2 for illustration). $\hat{\boldsymbol{\alpha}}_t = \{\hat{\alpha}_{t\hat{i}}\}_{\hat{i}=1,\ldots,\hat{L}}$

is the resized and normalized $\boldsymbol{\alpha}_t$ in order to ensure size consistency.

In some cases a phrase $\{y_t, \ldots, y_{t+l}\}$ refers to the same entity, therefore the individual words share the same attention region R_t. We define the phrase attention correctness as the maximum of the individual scores[1].

$$AC(\{y_t, \ldots, y_{t+l}\}) = \max(AC(y_t), \ldots, AC(y_{t+l})) \tag{5.13}$$

The intuition is that the phrase may contain some less interesting words whose attention map is ambiguous, and the attention maps of these words can be ignored by the max operation. For example, when evaluating the phrase "a group of people", we are more interested in the attention correctness for "people" rather than "of".

We discuss next how to find ground truth attention regions during testing, in order to apply this evaluation metric.

5.4.2 Ground Truth Attention Region During Testing

In order to compute attention correctness, we need the correspondence between regions in the image and phrases in the caption. However, in the testing stage, the generated caption is often different from the ground truth captions. This makes evaluation difficult, because we only have corresponding image regions for the phrases in the ground truth caption, but not *any* phrase. To this end, we propose two strategies.

Ground Truth Caption One option is to enforce the model to output the ground truth sentence by resetting the input to the ground truth word at each time step. This procedure to some extent allows us to "decorrelate" the attention module from the captioning component, and diagnose if the learned attention module is meaningful. Since the generated caption exactly matches the ground truth, we compute attention correctness for all noun

[1] In the experiments, we found that changing the definition from maximum to average does not affect our main conclusion.

phrases in the test set.

Generated Caption Another option is to align the entities in the generated caption to those in the ground truth caption. For each image, we first extract the noun phrases of the generated caption using a POS tagger (e.g. Stanford Parser [169]), and see if there exists a word-by-word match in the set of noun phrases in the ground truth captions. For example, if the generated caption is "A dog jumping over a hurdle" and one of the ground truth captions is "A cat jumping over a hurdle", we match the noun phrase "a hurdle" appearing in both sentences. We then calculate the attention correctness for the matched phrases only.

5.5 Experiments

5.5.1 Implementation Details

Implicit/Supervised Attention Models All implementation details strictly follow [272]. We resize the image such that the shorter side has 256 pixels, and then center crop the 224×224 image, before extracting the conv5_4 feature of the 19 layer version of VGG net [232] pretrained on ImageNet [47]. The model is trained using stochastic gradient descent with the Adam algorithm [125]. Dropout [238] is used as regularization. We use the hyperparameters provided in the publicly available code[2]. We set the number of LSTM units to 1300 for Flickr30k and 1800 for COCO.

Ground Truth Attention for Strong Supervision Model We experiment with our strong supervision model on the Flickr30k dataset [281]. The Flickr30k Entities dataset [203] is used for generating the ground truth attention maps. For each entity (noun phrase) in the caption, the Flickr30k Entities dataset provides the corresponding bounding box of the entity in the image. Therefore ideally, the model should "attend to" the marked region

[2] https://github.com/kelvinxu/arctic-captions

when predicting the associated words. We evaluate on noun phrases only, because for other types of words (e.g. determiner, preposition) the attention might be ambiguous and meaningless.

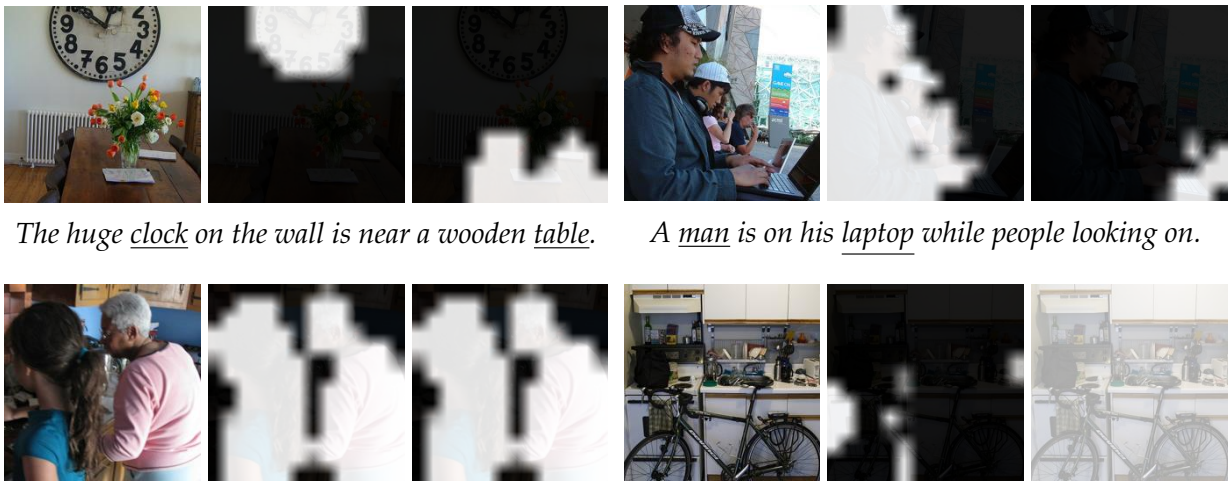
Ground Truth Attention for Weak Supervision Model The MS COCO dataset [148] contains instance segmentation masks of 80 classes in addition to the captions, which makes it suitable for our model with weak supervision. We only construct $\boldsymbol{\beta}_t$ for the nouns in the captions, which are extracted using the Stanford Parser [169]. The similarity function in Equation 5.11 is chosen to be the cosine distance between word vectors [180] pretrained on GoogleNews[3], and we set an empirical threshold of $1/3$ (i.e. only keep those with cosine distance greater than the threshold).

The $\boldsymbol{\beta}_t$ generated in this way still contains obvious errors, primarily because word2vec cannot distinguish well between objects and scenes. For example, the similarity between the word "kitchen" and the object class "spoon" is above threshold. But when generating a scene word like "kitchen", the model should be attending to the whole image instead of focusing on a small object like "spoon".

To address this problem, we refer to the supplement of [148], which provides a scene category list containing key words of scenes used when collecting the dataset. Whenever some word in this scene category list appears in the caption, we set $\boldsymbol{\beta}_t$ to be uniform, i.e. equal attention across image. This greatly improves the quality of $\boldsymbol{\beta}_t$ in some cases (see illustration in Figure 5.3).

Comparison of Metric Designs To show the legitimacy of our attention correctness metric, we compute the spearsman correlation of our design and three other metrics: negative L1 distance, negative L2 distance, and KL divergence between $\hat{\boldsymbol{\beta}}_t$ and $\hat{\boldsymbol{\alpha}}_t$. On the Flickr30k test set with implicit attention and ground truth caption, the spearsman

[3] https://code.google.com/archive/p/word2vec/



The huge clock on the wall is near a wooden table.

A man is on his laptop while people looking on.

A young girl and a woman preparing food in a kitchen.

A bicycle parked in a kitchen by the stove.

Figure 5.3. Ground truth attention maps generated for COCO. The first two examples show successful cases. The third example is a failed case where the proposed method aligns both "girl" and "woman" to the "person" category. The fourth example shows the necessity of using the scene category list. If we do not distinguish between object and scene (middle), the algorithm proposes to align the word "kitchen" with objects like "spoon" and "oven". We propose to use uniform attention (right) in these cases.

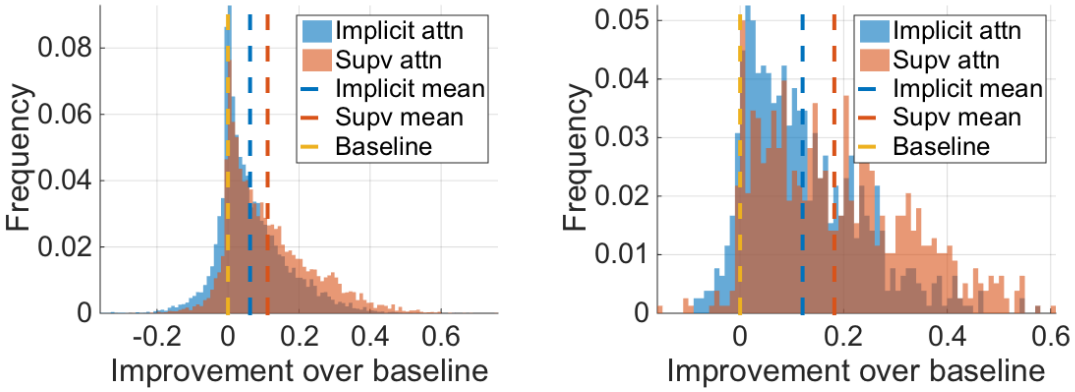
correlations between any two are all above 0.96, suggesting that all these measurements are similar. Therefore our metric statistically correlates well with other metrics, while being the most intuitive.

5.5.2 Evaluation of Attention Correctness

In this subsection, we quantitatively evaluate the attention correctness of both the implicit and the supervised attention model. All experiments are conducted on the 1000 test images of Flickr30k. We compare the result with a uniform baseline, which attends equally across the whole image. Therefore the baseline score is simply the size of the bounding box over the size of the whole image. The results are summarized in Table 5.1.

Table 5.1. Attention correctness and baseline on Flickr30k test set. Both the implicit and the (strongly) supervised models outperform the baseline. The supervised model performs better than the implicit model in both settings.

Caption	Model	Baseline	Correctness
Ground Truth	Implicit	0.3214	0.3836
	Supervised	0.3214	**0.4329**
Generated	Implicit	0.3995	0.5202
	Supervised	0.3968	**0.5787**



(a) Ground truth caption result

(b) Generated caption result

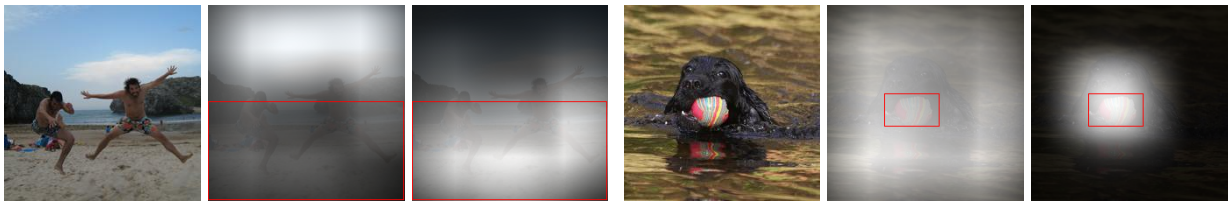
Figure 5.4. Histograms of attention correctness for the implicit model and the supervised model on the Flickr30k test set. The more to the right the better.

Ground Truth Caption Result In this setting, both the implicit and supervised models are forced to produce exactly the same captions, resulting in 14566 noun phrase matches. We discard those with no attention region or full image attention (as the match score will be 1 regardless of the attention map). For each of the remaining matches, we resize the original attention map from 14×14 to 224×224 and perform normalization before we compute the attention correctness for this noun phrase.

Both models are evaluated in Figure 5.4a. The horizontal axis is the improvement over baseline, therefore a better attention module should result in a distribution further to the right. On average, both models perform better than the baseline. Specifically, the

Girl rock climbing on the rock wall.

A young smiling child hold his toy alligator up to the camera.



Two male friends in swimming trunks jump on the beach while people in the background lay in the sand.

A black dog swims in water with a colorful ball in his mouth.

Figure 5.5. Attention correctness using ground truth captions. From left to right: original image, implicit attention, supervised attention. The red box marks correct attention region (from Flickr30k Entities). In general the attention maps generated by our supervised model have higher quality.

average gain over uniform attention baseline is 6.22% for the implicit attention model [272], and 11.14% for the supervised version. Visually, the distribution of the supervised model is further to the right. This indicates that although the implicit model has captured some aspects of attention, the model learned with strong supervision has a better attention module.

In Figure 5.5 we show some examples where the supervised model correctly recovers the spatial location of the underlined entity, while the implicit model attends to the wrong region.

Generated Caption Result In this experiment, word-by-word match is able to align 909 noun phrases for the implicit model and 901 for the supervised version. Since this strategy is rather conservative, these alignments are correct and reliable, as verified by a manual check. Similarly, we discard those with no attention region or full image attention, and

Table 5.2. Attention correctness and baseline on the Flickr30k test set (generated caption, same matches for implicit and supervised) with respect to bounding box size. The improvement is greatest for small objects.

BBox Size	Model	Baseline	Correctness
Small	Implicit	0.1196	0.2484
	Supervised	0.1196	**0.3682**
Medium	Implicit	0.3731	0.5371
	Supervised	0.3731	**0.6117**
Large	Implicit	0.7358	0.8117
	Supervised	0.7358	**0.8255**

perform resize and normalization before we compute the correctness score.

The results are shown in Figure 5.4b. In general the conclusion is the same: the supervised attention model produces attention maps that are more consistent with human judgment. The average improvement over the uniform baseline is 12.07% for the implicit model and 18.19% for the supervised model, which is a 50% relative gain.

In order to diagnose the relationship between object size and attention correctness, we further split the test set equally with small, medium, and large ground truth bounding box, and report the baseline and attention correctness individually. We can see from Table 5.2 that the improvement of our supervised model over the implicit model is greatest for small objects, and pinpointing small objects is stronger evidence of image understanding than large objects.

In Figure 5.6 we provide some qualitative results. These examples show that for the same entity, the supervised model produces more human-like attention than the implicit model.

5.5.3 Evaluation of Captioning Performance

We have shown that supervised attention models achieve higher attention correctness than implicit attention models. Although this is meaningful in tasks such as region grounding,

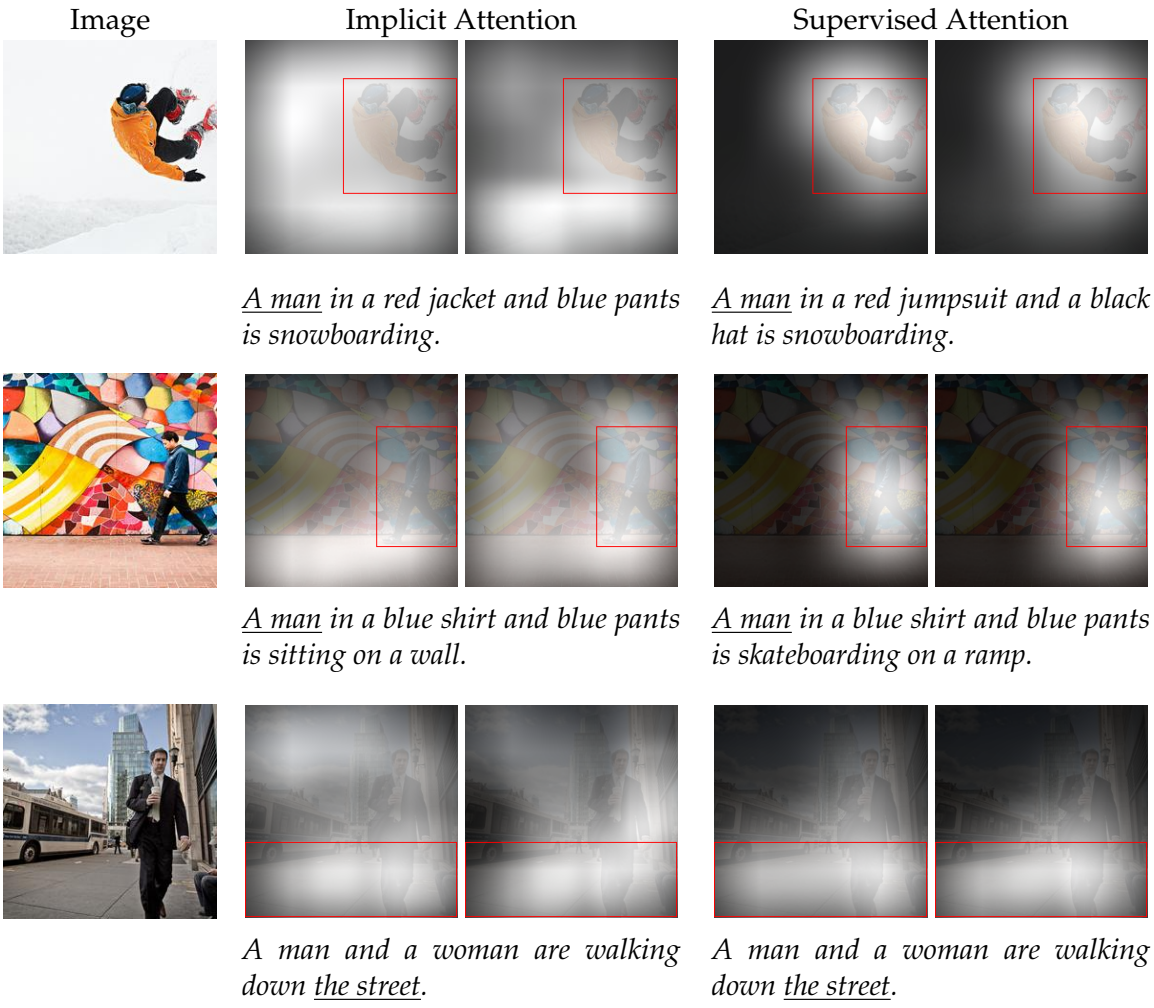


Figure 5.6. Attention correctness using generated captions. The red box marks correct attention region (from Flickr30k Entities). We show two attention maps for the two words in a phrase. In general the attention maps generated by our supervised model have higher quality.

in many tasks attention only serves as an intermediate step. We may be more interested in whether supervised attention model also has better captioning performance, which is the end goal. The intuition is that a meaningful dynamic weighting of the input vectors will allow later components to decode information more easily. In this subsection we give experimental support.

We report BLEU [198] and METEOR [14] scores to allow comparison with [272]. In Table 5.3 we show both the scores reported in [272] and our implementation. Note that

Table 5.3. Comparison of image captioning performance. * indicates our implementation. Caption quality consistently increases with supervision, whether it is strong or weak.

Dataset	Model	BLEU-3	BLEU-4	METEOR
Flickr30k	Implicit	28.8	19.1	18.49
	Implicit*	29.2	20.1	19.10
	Strong Sup	**30.2**	**21.0**	**19.21**
COCO	Implicit	34.4	24.3	23.90
	Implicit*	36.4	26.9	24.46
	Weak Sup	**37.2**	**27.6**	**24.78**

Table 5.4. Captioning scores on the Flickr30k test set for different attention correctness levels in the generated caption, implicit attention experiment. Higher attention correctness results in better captioning performance.

Correctness	BLEU-3	BLEU-4	METEOR
High	38.0	28.1	23.01
Middle	36.5	26.1	21.94
Low	35.8	25.4	21.14

our implementation of [272] gives slightly improved result over what they reported. We observe that BLEU and METEOR scores consistently increase after we introduce supervised attention for both Flickr30k and COCO. Specifically in terms of BLEU-4, we observe a significant increase of 0.9 and 0.7 percent respectively.

To show the positive correlation between attention correctness and caption quality, we further split the Flickr30k test set (excluding those with zero alignment) equally into three sets with high, middle, and low attention correctness. The BLEU-4 scores are 28.1, 26.1, 25.4, and METEOR are 23.01, 21.94, 21.14 respectively (see Table 5.4). This indicates that higher attention correctness means better captioning performance.

5.6 Discussion

In this work we make a first attempt to give a quantitative answer to the question: to what extent are attention maps consistent with human perceptions? We first define attention correctness in terms of consistency with human annotation at both the word level and phrase level. In the context of image captioning, we evaluated the state-of-the-art models with implicitly trained attention modules. The quantitative results suggest that although the implicit models outperform the uniform attention baseline, they still have room for improvement.

We then show that by introducing supervision of attention map, we can improve both the image captioning performance and attention map quality. In fact, we observe a positive correlation between attention correctness and captioning quality. Even when the ground truth attention is unavailable, we are still able to utilize the segmentation masks with object category as a weak supervision to the attention maps, and significantly boost captioning performance.

We believe closing the gap between machine attention and human perception is necessary, and expect to see similar efforts in related fields.

Chapter 6

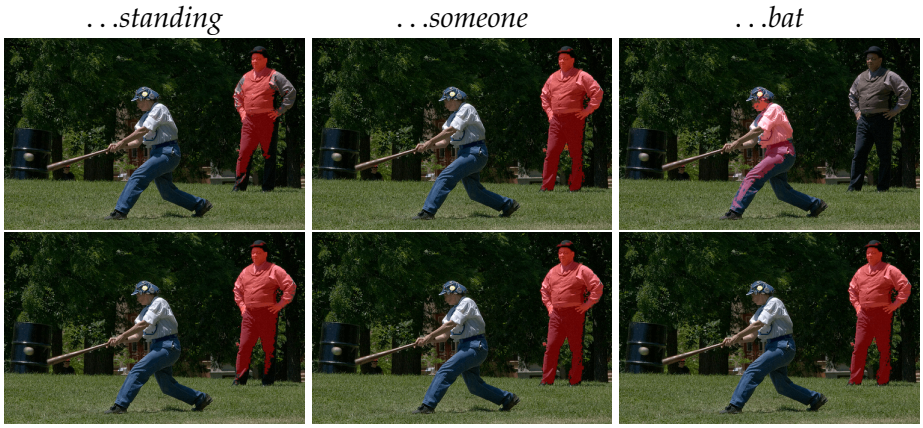
Recurrent Multimodal Interaction for Referring Image Segmentation

This chapter describes an improved model for referring expression comprehension, which also facilitates diagnosis with its intermediate outputs.

6.1 Introduction

In this chapter, we study the challenging problem of using natural language expressions to segment an image. Given both an image and a natural language expression, we are interested in segmenting out the corresponding region referred by the expression. This problem was only introduced recently, but has great value as it provides new means for interactive image segmentation. Specifically, people can segment/select image regions of their interest by typing natural language descriptions or even speaking to the computer [139].

Given the success of convolutional neural networks in semantic segmentation [29], [30], [161], an immediate way to tackle this problem is to augment the convolutional semantic segmentation networks with a LSTM [92] sentence encoder [100], so that the image features and sentence representation can be combined to produce the desired mask. In fact, this sentence-to-image interaction scheme has been also adopted by recent methods on referring object localization [295] and visual question answering tasks [5].



Man in a vest and blue jeans standing watching someone swing a bat.

Figure 6.1. Given the image and the referring expression, we are interested in segmenting out the referred region. Each column shows segmentation result until after reading the underlined word. Our model (second row) explicitly learns the progression of multimodal interaction with convolutional LSTM, which helps long-term memorization and correctly segments out the referred region compared with the baseline model (first row) which uses language-only LSTM.

However, this sentence-to-image scheme does not reflect how humans tackle this problem. In sentence-picture verification, it is found through eye tracking that when pictures and sentences are presented together, people either follow a image-sentence-image reading sequence, or go back-and-forth between sentence and picture a number of times before making the decision [251]. In other words, the interaction between image and sentence should prevail from the beginning to the end of the sentence, instead of only happening at the end of the sentence. Presumably this is because the semantic information is more concrete and therefore more easily remembered when grounded onto the image. For example, consider the expression "the man on the right wearing blue". Without seeing an actual image, all information in the sentence needs to be remembered, meaning the sentence embedding needs to encode `IS_MAN`, `ON_RIGHT`, `WEAR_BLUE` jointly. However, with the actual image available, the reasoning process can be decomposed as a sequential process, where the model first identifies all pixels that agree with `IS_MAN`,

then prunes out those that do not correspond with ON_RIGHT, and finally suppresses those that do not agree with WEAR_BLUE.

Motivated by this sequential decision making theory, we propose a two-layered convolutional multimodal LSTM network that explicitly models word-to-image interaction. Different from the language-only LSTM encoder in previous works [100], the convolutional multimodal LSTM takes both visual feature and language representation as input to generate the hidden state that retains both the spatial and semantic information in memory. Therefore its hidden state models how the multimodal feature progresses over time or word-reading order. After seeing the last word, we use a convolution layer to generate the image segmentation result.

In summary, the contribution of this chapter is three-fold:

- We propose a novel model, namely convolutional multimodal LSTM, to encode the sequential interactions between individual semantic, visual, and spatial information.
- We demonstrate the superior performance of the word-to-image multimodal LSTM approach on benchmark datasets over the baseline model.
- We analyze the intermediate output of the proposed multimodal LSTM approach and empirically explain how this approach enforces a more effective word-to-image interaction.

6.2 Related Work

In this section, we review recent studies that are tightly related to our work in the following three areas: semantic segmentation, referring expression localization, and multimodal interaction representation.

Semantic Segmentation Many state-of-the-art semantic segmentation models employ a fully convolutional network [161] architecture. FCN converts the fully connected layers in

VGG network [232] into convolutional layers, thereby allowing dense (although downsampled) per-pixel labeling. However, too much downsampling (caused by pooling layers in the VGG architecture) prohibits the network from generating high quality segmentation results. DeepLab [29] alleviates this issue by discarding two pooling operations with atrous convolution. With Residual network [89] as its backbone architecture, DeepLab [30] is one of the leading models on Pascal VOC [61]. We use both ResNet-101 (with atrous convolution) and DeepLab ResNet-101 to extract image features in a fully convolutional manner. Following [29], [30], we also report the result of using DenseCRF [130] for refinement.

Referring Expression Localization More and more interest arise recently in the problem of localizing objects based on a natural language expression. In [170] and [102], image captioning models [53], [171] are modified to score the region proposals, and the one with the highest score is considered as the localization result. In [212], the alignment between the description and image region is learned by reconstruction with attention mechanism. [283] improved upon [170] by explicitly handling objects of the same class within the same image, while [185] focused on discovering interactions between the object and its context using multiple-instance learning. However all these works aim at finding a bounding box of the target object instead of segmentation mask. Perhaps the most relevant work to ours is [100], which studies the same problem of image segmentation based on referring expressions. Our approach differs in that we model the *sequential* property of interaction between natural language, visual, and spatial information. In particular, we update the segmentation belief after seeing each word.

Multimodal Interaction Representation Our work is also related to multimodal feature fusion in visual question answering [69], [124], [168] and image captioning [53]. In [53] the input to LSTM is the image feature and the previous word's embedding, whereas in [168] the input to LSTM is the image feature and individual question word's embedding.

Attention mechanism [149], [153], [272], [277], [278] may also be applied, mostly to improve the relevance of image features. In both tasks the goal is to generate a textual sequence. Here instead, we use the LSTM hidden states to generate segmentation, which is not commonly considered a sequential task and requires preservation of spatial location. We achieve this by applying LSTM in a convolutional manner [40], [65], [225], unlike prior work on recurrent attention [136], [182].

6.3 Models

In this section, we first introduce our notation for this problem (Section 6.3.1), and then describe the baseline model based on the sentence-to-image scheme [100] (Section 6.3.2), which only models the progression of semantics. In Section 6.3.3 we propose convolutional multimodal LSTM for fusing both modalities and model the progression of multimodal features in addition to the progression of semantics.

6.3.1 Notation

In the referring image segmentation problem, we are given both an image I and a natural language description $S = \{w_1, w_2, \ldots, w_T\}$, where w_t ($t \in \{1, 2, \ldots, T\}$) are individual words in the sentence. The goal is to segment out the corresponding region in the image. We will use R for prediction and $\hat{R}$ for ground truth. $R^{ij} \in (0, 1)$ represents the foreground probability of a pixel, where i and j are spatial coordinates. $\hat{R}^{ij} \in \{0, 1\}$, where 1 means the pixel is referred to by S and 0 otherwise.

6.3.2 Baseline Model

Our model is based on the model proposed in [100]. In [100], given an image of size $W \times H$, an FCN-32s [161] is used to extract image features with size $W' \times H' \times D_I$, where $W' = W/32$ and $H' = H/32$. The image features are then concatenated with spatial

coordinates to produce a $W' \times H' \times (D_I + 8)$ tensor. The 8 spatial coordinate dimensions follow the implementation of [100]. The normalized horizontal/vertical position uses 3 dimensions each. The remaining 2 dimensions are $1/W'$ and $1/H'$. We use $\mathbf{v}^{ij} \in \mathbb{R}^{D_I+8}$ to represent the image-spatial feature at a specific spatial location.

As for the referring expression, every word w_t is one-hot encoded and mapped to a word embedding $\mathbf{w}_t$. The entire sentence is then encoded with an LSTM into a vector $\mathbf{h}_T$ of size D_S, where $\mathbf{h}_t$ represents the hidden state of LSTM at time step t:

$$\text{LSTM} : (\mathbf{w}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}) \rightarrow (\mathbf{h}_t, \mathbf{c}_t) \tag{6.1}$$

$$\begin{pmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} M_{4n, D_S+n} \begin{pmatrix} \mathbf{w}_t \\ \mathbf{h}_{t-1} \end{pmatrix} \tag{6.2}$$

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \mathbf{g} \tag{6.3}$$

$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t) \tag{6.4}$$

where n is the size of the LSTM cell. $\mathbf{i}, \mathbf{f}, \mathbf{o}, \mathbf{g}$ are the input gates, forget gates, output gets, and memory gates respectively. $\mathbf{c}_t$ are the memory states at time step t.

The vector $\mathbf{h}_T$ is then concatenated with the image features and spatial coordinates at all locations to produce a $W' \times H' \times (D_I + D_S + 8)$ tensor. Two additional convolutional layers and one deconvolution layer are attached to the tensor to produce the final segmentation mask $R \in \mathbb{R}^{W \times H}$.

Given the ground truth binary segmentation mask $\hat{R}$, the loss function is

$$L_{high} = \frac{1}{WH} \sum_{i=1}^{W} \sum_{j=1}^{H} \Big(\hat{R}^{ij} * -\log(R^{ij}) + (1 - \hat{R}^{ij}) * -\log(1 - R^{ij}) \Big) \tag{6.5}$$

The whole network is trained with standard back-propagation.

Our baseline employs the same architecture, except that we use ResNet-101 [89] instead

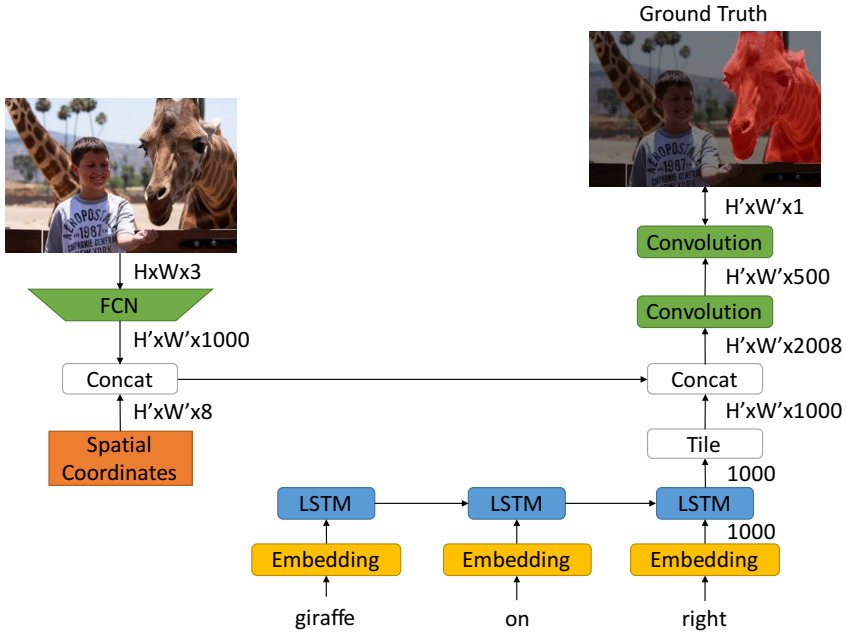


Figure 6.2. Network architecture of the baseline model described in Section 6.3.2. In this model, the entire sentence is encoded into a fixed vector with language-only LSTM without using visual information.

of FCN-32s to extract image features. One limitation of FCN-32s is that downsampling by 32 makes W' and H' too small. Therefore similar to the treatment of DeepLab [29], [30], we reduce the stride of conv4_1 and conv5_1 in ResNet-101 from 2 to 1, and use atrous convolution of rate 2 and 4 to compensate for the change. This operation reduces the downsampling rate from 32 to 8, which is relatively dense and allows loss to be computed at the feature resolution ($W' = W/8, H' = H/8$) instead of the image resolution. Therefore in our model, the loss function becomes

$$L_{low} = \frac{1}{W'H'} \sum_{i=1}^{W'} \sum_{j=1}^{H'} \Big(\hat{R}^{ij} * -\log(R^{ij}) + (1 - \hat{R}^{ij}) * -\log(1 - R^{ij}) \Big) \tag{6.6}$$

We use bilinear interpolation to upsample $R \in \mathbb{R}^{W' \times H'}$ at test time.

We are going to show in the experimental section that combining ResNet with atrous

convolution results in a more competitive baseline model and easier training procedure.

6.3.3 Recurrent Multimodal Interaction Model

In the baseline model described above, segmentation is performed once, after the model has seen and memorized the entire referring expression. The memorization is the process of updating LSTM hidden states while scanning the words in the expression one by one. However, as discussed earlier, this requires the model to memorize all the attributes in the sentence jointly. We instead utilize the sequential property of natural language and turn referring image segmentation into a sequential process. This requires the language model to have access to the image from the beginning of the expression, allowing the semantics to be grounded onto the image early on. Therefore we consider modeling of the multimodal interaction, i.e. a scheme that can memorize the *multimodal* information (language, image, spatial information, and their interaction), which has direct influence on the segmentation prediction.

We use a multimodal LSTM to capture the progression of rich multimodal information through time as shown in Figure 6.3. Specifically, a multimodal LSTM (mLSTM) uses the concatenation of the language representation $\mathbf{l}_t \in \mathbb{R}^{D_S}$ and the visual feature at a specific spatial location $\mathbf{v}_{ij} \in \mathbb{R}^{D_I+8}$ as its input vector:

$$\text{mLSTM} : (\begin{bmatrix}\mathbf{l}_t \\ \mathbf{v}^{ij}\end{bmatrix}, \mathbf{h}^{ij}_{t-1}, \mathbf{c}^{ij}_{t-1}) \rightarrow (\mathbf{h}^{ij}_t, \mathbf{c}^{ij}_t) \tag{6.7}$$

The same mLSTM operation is shared for all image locations. This is equivalent to treating the mLSTM as a 1×1 convolution over the feature map of size $W' \times H' \times (D_I + D_S + 8)$. In other words, this is a convolutional LSTM that shares weights both across spatial location and time step.

The baseline model uses language-only LSTM (Equation 6.1) to encode the referring expression, and concatenate it with the visual feature to produce $\begin{bmatrix}\mathbf{h}_T \\ \mathbf{v}^{ij}\end{bmatrix}$. One advantage of multimodal LSTM is that either of the two components can be produced by it. The matrix

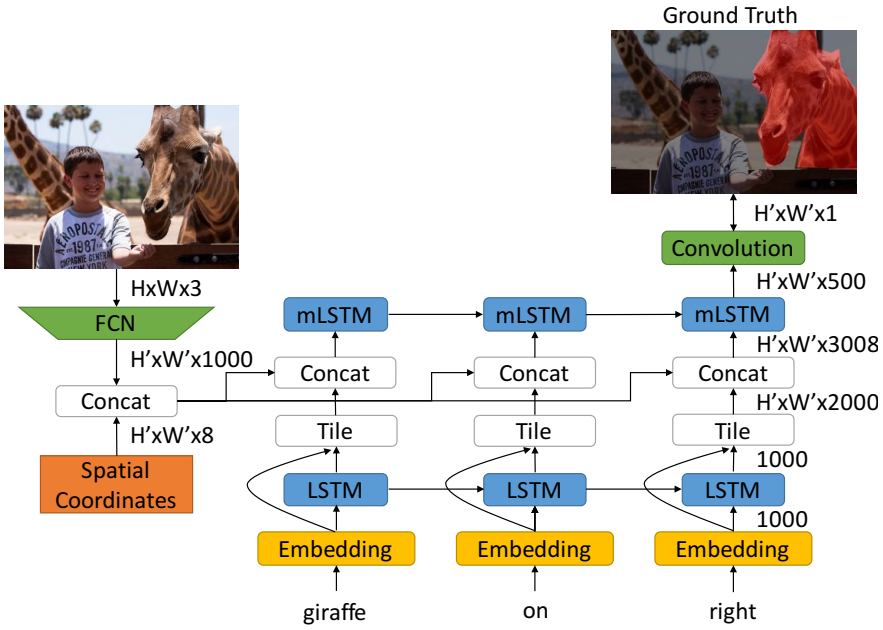


Figure 6.3. Network architecture of the RMI model described in Section 6.3.3. By using the convolutional multimodal LSTM, our model allows multimodal interaction between language, image, and spatial information at each word. The mLSTM is applied to all location in the image and implemented as a 1×1 convolution.

M in multimodal LSTM will be of size $4n \times (D_S + D_I + 8 + n)$. If $M_{1:4n, D_S+1:D_S+D_I+8} = \mathbf{0}$, then the mLSTM will essentially ignore the visual part of the input, and encode only the semantic information. On the other hand, if the mLSTM ignores the language representation, the mLSTM will see the same input $\mathbf{v}_{ij}$ at all time steps, therefore very likely to retain that information.

From another perspective, multimodal LSTM forces word-visual interaction and generates multimodal feature at every recurrent step, which is key to good segmentation. In the baseline model, in order for the language representation to reach the multimodal level, it has to go through all subsequent LSTM cells as well as a convolution layer:

$$\mathbf{l}_t \xrightarrow{LSTM} \mathbf{h}_T \xrightarrow{Concat} \begin{bmatrix} \mathbf{h}_T \\ \mathbf{v}^{ij} \end{bmatrix} \xrightarrow{Conv} \text{multimodal feature} \tag{6.8}$$

while with multimodal LSTM this can be done with just the (multimodal) LSTM cells:

$$\mathbf{l}_t \xrightarrow{Concat} \begin{bmatrix} \mathbf{l}_t \\ \mathbf{v}^{ij} \end{bmatrix} \xrightarrow{mLSTM} \text{multimodal feature} \tag{6.9}$$

Note that the visual feature still only needs one weight layer to become multimodal.

In our Recurrent Multimodal Interaction (RMI) model, we take the language representation $\mathbf{l}_t$ to be the concatenation of language-only LSTM hidden state in Equation 6.1 and word embedding $\begin{bmatrix} \mathbf{h}_t \\ \mathbf{w}_t \end{bmatrix}$. This forms a two-layer LSTM structure, where the lower LSTM only encodes the semantic information, while the upper LSTM generates the multimodal feature. The lower language-only LSTM is spatial-agnostic, while the upper multimodal LSTM preserves feature resolution $H' \times W'$.

6.4 Experiments

6.4.1 Datasets

We use four datasets to evaluate our model: Google-Ref [170], UNC [283], UNC+ [283], and ReferItGame [123].

Google-Ref contains 104560 expressions referring to 54822 objects from 26711 images selected from MS COCO [148]. These images all contain 2 to 4 objects of the same type. In general the expressions are longer and with richer descriptions, with an average length of 8.43 words. Although the dataset has primarily been used for referring object detection [170], [185], [283], where the goal is to return a bounding box of the referred object, it is also suitable for referring image segmentation, since the original MS COCO annotation contains segmentation masks. We use the same data split as [170].

UNC and UNC+ are also based on MS COCO images. Different from Google-Ref, these two datasets are collected interactively in a two-player game [123]. The difference between the two datasets is in UNC no restrictions are enforced on the referring expression, while in UNC+ no location words are allowed in the expression, meaning the annotator has to

describe the object purely by its appearance. UNC consists of 142209 referring expressions for 50000 objects in 19994 images, and UNC+ consists of 141564 expressions for 49856 objects in 19992 images. We use the same data split as [283].

ReferItGame contains 130525 expressions referring to 96654 distinct objects in 19894 natural images. Different from the other three datasets, ReferItGame contains "stuff" segmentation masks, such as "sky" and "water", in addition to objects. In general the expressions are shorter and more concise, probably due to the collection process as a two-player game. We use the same data split as [100].

6.4.2 Implementation Details

[100], [101] both use the VGG network [232] pretrained on ImageNet [47] as visual feature extractor. We instead experiment with two alternatives: ResNet-101 pretrained on ImageNet, and DeepLab-101 finetuned on Pascal VOC [61].

In our experiments, we resize (while keeping aspect ratio) and pad (with zero) all images and ground truth segmentation to $W \times H$, and in all our experiments $W = H = 320$. As for the feature resolution $W' = H' = 40$. The image feature has dimension $D_I = 1000$, and the sentence vector has dimension $D_S = 1000$. We choose the cell size of mLSTM to be 500. For referring expressions of length more than 20, we only keep the first 20 words. All architecture details are in Figure 6.2 6.3, where sizes of blobs are marked.

In [100] a three-stage training strategy is used. A detection network is first trained, which is used to initialize the low resolution version of the model. After training the low resolution version with $W' = H' = 16$, it is again used to initialize the high resolution version, where a deconvolution layer is learned. We instead only train once using the loss function defined in Equation 6.6, and observe fast convergence. This is probably due to the higher spatial resolution allowed by atrous convolution. We use the Adam [125] optimizer with a fixed learning rate of 0.00025. We set the batch size to 1 and weight decay to 0.0005.

Table 6.1. Comparison of segmentation performance (IOU). In the first column, R means ResNet weights, D means DeepLab weights, and DCRF means DenseCRF.

	Google-Ref	UNC			UNC+			ReferItGame
	val	val	testA	testB	val	testA	testB	test
[100], [101]	28.14	-	-	-	-	-	-	48.03
R+LSTM	28.60	38.74	39.18	39.01	26.25	26.95	24.57	54.01
R+RMI	**32.06**	**39.74**	**39.99**	**40.44**	**27.85**	**28.69**	**26.65**	**54.55**
R+LSTM+DCRF	28.94	39.88	40.44	40.07	26.29	27.03	24.44	55.90
R+RMI+DCRF	**32.85**	**41.17**	**41.35**	**41.87**	**28.26**	**29.16**	**26.86**	**56.61**
D+LSTM	33.08	43.27	43.60	43.31	28.42	28.57	27.70	56.83
D+RMI	**34.40**	**44.33**	**44.74**	**44.63**	**29.91**	**30.37**	**29.43**	**57.34**
D+LSTM+DCRF	33.11	43.97	44.25	44.07	28.07	28.29	27.44	58.20
D+RMI+DCRF	**34.52**	**45.18**	**45.69**	**45.57**	**29.86**	**30.48**	**29.50**	**58.73**

We evaluate using two metrics: Precision@X ($X \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$) and Intersection over Union (IOU), where Precision@X means the percentage of images with IOU higher than X. This is consistent with previous work [100], [101] to allow for comparison. Here we report the most indicative IOU.

In addition to evaluating the direct segmentation output, we also report results after applying DenseCRF [130] for refinement. We use the same hyperparameters used in [30].

6.4.3 Quantitative Results

The segmentation performance (IOU) on all datasets are summarized in Table 6.1.

We first observe that the performance consistently increases by replacing the VGG-based FCN-32s with ResNet. This indicates that ResNet can provide better image features for segmentation purpose, which likely comes from both stronger network and higher spatial resolution. DeepLab delivers even higher baseline since its weights have been finetuned on segmentation datasets, which makes the knowledge transfer easier.

We then study the effect of mLSTM. Our RMI models with mLSTM consistently out-

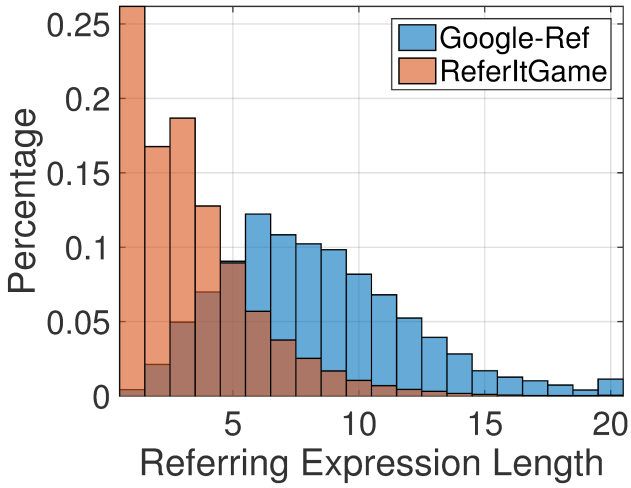


Figure 6.4. The distribution of referring expression length in the Google-Ref and ReferItGame test set. Most of the referring expressions in ReferItGame are short, with over 25 percent single word description. The distributions of UNC and UNC+ are very similar to that of ReferItGame since the data collection method is the same.

perform those with language-only LSTM by a large margin regardless of the image feature extractor and dataset. This shows that mLSTM can successfully generate multimodal features that improve segmentation. Specifically, on the Google-Ref dataset using ResNet weights, we observe an IOU increase of nearly 3.5% over the baseline model.

By comparison, the performance increase using mLSTM is not as high on ReferItGame. One reason is that the dataset is easier as indicated by the metrics (over 20 percent higher IOU than Google-Ref), and the baseline model already performs well. Another reason is that the descriptions in this dataset are in general much shorter (see Figure 6.4), and as a result sequential modeling does not have as much effect. In fact, over 25 percent images in the ReferItGame test set only has one word as its description.

Another interesting observation is that the performance is considerably worse on UNC+ than on UNC (over 10 percent IOU difference). As aforementioned, the only difference between the two datasets is in UNC+ there is no spatial/location indicator words that the model can utilize, and the model must understand the semantics in order to output correct segmentation. This suggests that the LSTM language encoder may be the main barrier in

Table 6.2. IOU performance break-down on Google-Ref.

Length	**1-5**	**6-7**	**8-10**	**11-20**
R + LSTM	32.29	28.27	27.33	26.61
R + RMI	35.34	31.76	30.66	30.56
Relative Gain	9.44%	12.37%	12.17%	14.81%

Table 6.3. IOU performance break-down on UNC.

Length	**1-2**	**3**	**4-5**	**6-20**
R + LSTM	43.66	40.60	33.98	24.91
R + RMI	44.51	41.86	35.05	25.95
Relative Gain	1.94%	3.10%	3.15%	4.19%

referring image segmentation performance.

We further show the advantage of our mLSTM model in sequential modeling by breaking down the IOU performance. Specifically, we want to study the relationship between IOU and referring expression length. To this end, we split the test set into 4 groups of increasing referring expression length with roughly equal size, and report the individual IOU on these groups. The results are summarized in Table 6.2 6.3 6.4 6.5. Our RMI model outperforms the baseline model in every group. More interestingly, the relative gain of our RMI model over the baseline model in general increases with the length of the referring expression. This suggests that mLSTM is better at fusing features over longer sequences, which we will also verify visually.

Finally, by applying the DenseCRF, we observe consistent improvement in terms of IOU. In addition, the IOU improvement on our RMI model is usually greater than the IOU improvement on the baseline model, suggesting that our model has better localization ability.

Table 6.4. IOU performance break-down on UNC+.

Length	1-2	3	4-5	6-20
R + LSTM	34.40	24.04	19.31	12.30
R + RMI	35.72	25.41	21.73	14.37
Relative Gain	3.84%	5.67%	12.55%	16.85%

Table 6.5. IOU performance break-down on ReferItGame.

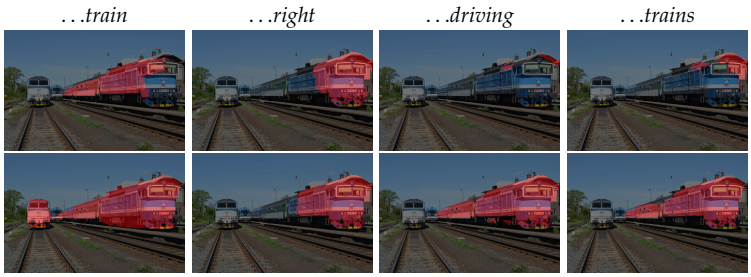
Length	1	2	3-4	5-20
R + LSTM	67.64	52.26	44.87	33.81
R + RMI	68.11	52.73	45.69	34.53
Relative Gain	0.69%	0.90%	1.82%	2.10%

6.4.4 Qualitative Results

As aforementioned, our RMI model is better than the baseline model in modeling long sequences. We can see from the examples in Figure 6.5 that the language-only LSTM is more easily distracted by words at the end of the sentence, resulting in unsatisfactory segmentation, while our model remains unaffected.

We suspect the reason is because our model can turn segmentation into a sequential process, saving the burden on the LSTM hidden state to encode the entire sentence. We are therefore interested in visualizing how the multimodal LSTM hidden state progresses over time. Each mLSTM hidden state is a feature tensor of size $H' \times W' \times 500$. We visualize this tensor by first doing bilinear interpolation and then collapsing the feature dimension via meanpooling, generating a $H \times W$ response map. We provide three examples in Figure 6.6. In the first example, after reading only "The bottom", the model is not sure about what objects is to be referred, and pays general attention to the bottom half of the image. As soon as it reads "luggage", the response map pinpoints the objects, and remembers the information until the end of the sentence to generate the correct segmentation output. In the second example, in the beginning after reading "The", the model appears unsure. After

A girl in white holding a Wii remote.



Blue train on the far right trail driving ahead of two other trains.

Dog close to the tall table.

Figure 6.5. Comparison of D+LSTM+DCRF (first row) and D+RMI+DCRF (second row). Each column shows segmentation result until after reading the underlined word.

reading "The small vase", it discards the largest vase and focuses on the other two. As soon as the language mentions "middle", the response in the middle is enhanced, and retained till the end of the expression. In the third example there is no location words, but the response around the correct region gradually enhances with "leather" and "chair", and the response on people is gradually suppressed after reading more words. We can see

Figure 6.6. Visualizing and understanding convolutional multimodal LSTM in our RMI model. The first column is the original image, and the last column is the final segmentation output of D+RMI+DCRF. The middle columns visualize the output of mLSTM at underlined words by meanpooling the 500-dimensional feature.

that the mLSTM is successful at learning meaningful multimodal feature interaction in a sequential fashion that is consistent with our intuition in the introduction section. The meaningful multimodal features make it easier for the last convolution layer to do binary segmentation.

In Figure 6.7 we provide some qualitative results of referring image segmentation on the four datasets. For Google-Ref, the language understanding is more challenging. In addition to handling longer sequences, it also needs to cope with all kinds of high level reasoning, e.g. "turning around a corner", and potentially redundant information, e.g. "listening to his music". For UNC, the expression is much shorter, and spatial words are allowed, e.g. "on left". For UNC+, the expression is more challenging. The image region could have just been described as "boy on right", but instead the model needs to reason

Figure 6.7. Qualitative results of referring image segmentation. From top down are images from Google-Ref, UNC, UNC+, ReferItGame respectively.

from attributes like "strip shirt" and "eyes closed". For ReferItGame, the segmentation target is more flexible as it contains "stuff" segments in addition to objects. We show that by propagating the multimodal feature, our RMI model can better keep the intermediate belief, usually resulting in a more complete segmentation result. The effect of DenseCRF is also clearly demonstrated. For example, for the first image, DenseCRF can better refine the D+RMI result to align the prediction to the edges, and for the third image, DenseCRF can suppress the scattered wrong prediction in the D+LSTM result.

6.5 Conclusion

In this work we study the challenging problem of referring image segmentation. Learning a good multimodal representation is essential in this problem, since segmentation represents the correspondence or consistency between images and language. Unlike previous work, which encodes the referring expression and image into vector representation independently, we build on the observation that referring image segmentation is a sequential process, and perform multimodal feature fusion after seeing every word in the referring expression. To this end we propose the Recurrent Multimodal Interaction model, a novel two-layer recurrent architecture that encodes the sequential interactions between individual words, visual information, and spatial information as its hidden state.

We show the advantage of our word-to-image scheme over the sentence-to-image scheme. Our model achieves the new state-of-the-art on all large-scale benchmark datasets. In addition, we visualize the mLSTM hidden state and show that the learned multimodal feature is human-interpretable and facilitates segmentation. In the future we plan to introduce more structure in language understanding.

Chapter 7

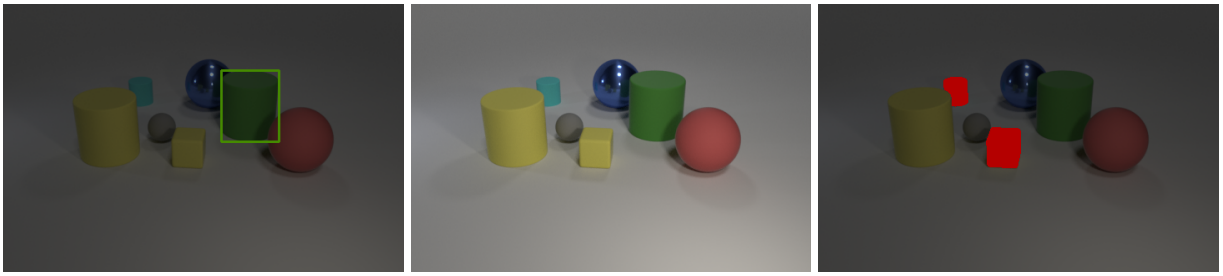
CLEVR-Ref+: Diagnosing Visual Reasoning with Referring Expressions

This chapter describes a diagnostic dataset for referring expressions, as well as a high-performing model which can naturally reveal its visual reasoning process.

7.1 Introduction

There has been significant research interest in the joint understanding of vision and natural language. While image captioning [53], [120], [153], [171] focuses on generating a sentence with image being the only input, visual question answering (VQA) [5], [71], [295] and referring expressions (REF) [102], [170] require comprehending both an image and a sentence, before generating an output. In this chapter, we focus on referring expressions, which is to identify the particular objects (in the form of segmentation mask or bounding box) in a given scene from natural language.

In order to study referring expressions, various datasets have been proposed [123], [170], [283]. These are real-world images annotated by crowdsource workers. The advantage of these datasets is that they, to a certain extent, reflect the complexity and nuances of the real world. Yet inevitably, they also have limitations. First, they usually exhibit strong biases that may be exploited by the models [41]. Roughly speaking, this means simply selecting the salient foreground object (i.e., discarding the referring expression) will yield a much



The big thing(s) that are behind the second one of the big thing(s) from front and to the right of the first one of the large sphere(s) from left

Any other things that are the same size as the fifth one of the thing(s) from right

Figure 7.1. Examples from our CLEVR-Ref+ dataset. We use the same scenes as those provided in CLEVR [112]. Instead of asking questions about the scene, we ask the model to either return one bounding box (as illustrated on the left) or return a segmentation mask (could potentially be multiple objects; illustrated on the right) based on the given referring expression.

higher baseline than random. This casts doubts on the true level of understanding within current REF models. Second, evaluation can only be conducted on the final segmentation mask or bounding box, but not the intermediate step-by-step reasoning process. For example, for the referring expression "Woman to the left of the red suitcase", a reasonable reasoning process should be first find all suitcases in the image, then identify the red one among them, finally segment the woman to its left. Clearly this requires significantly more high-quality annotations, which are currently unavailable and hard to collect.

To address these concerns and echo similar efforts in visual question answering (i.e., CLEVR [112]), we propose CLEVR-Ref+, a synthetic diagnostic dataset for referring expressions. The advantage of using a synthetic dataset is that we have full control over the scene, and dataset bias can be minimized by employing a uniform sampling strategy. Also, the referring expressions are now automatically annotated with the true underlying reasoning process, so a step-by-step analysis becomes much more plausible.

We make much effort in constructing CLEVR-Ref+ to make sure it is well adapted and applicable to the referring expression task. First, we turn the original questions in CLEVR into their corresponding referring expression format. Second, we change the

output space from textual answers (in the form of a word) to referred objects (in the form of segmentation mask or bounding box). Third, we analyzed statistics from real-world REF datasets and found that there are some common types of referring expressions (e.g., "The second sphere from left") that are not included in CLEVR templates. In our CLEVR-Ref+, we add support for these types of expressions to better match the variety of referring expressions used in real world.

We tested several state-of-the-art referring expression models on our CLEVR-Ref+ dataset. This includes both those designed for referring segmentation [152] and detection [282], [284]. In addition to evaluating the overall IoU and accuracy as previous datasets, we can now do a more detailed breakdown and analysis in terms of sub-categories. For example, we found that it is especially hard for the models to understand ordinality. This could point to important research directions in the future.

Besides diagnosing these existing models, we also propose IEP-Ref, a Neural Module Network [4] solution based on IEP [113]. Experiments show that the IEP-Ref model achieved excellent performance on CLEVR-Ref+ with its explicit, step-by-step functional program and module network execution engine, suggesting the importance of compositionality. Very interestingly, we found that the module trained on translating the last module output to segmentation mask is general, and can produce excellent human-interpretable segmentation masks when attached to intermediate module outputs, revealing the entire reasoning process. We believe ours is the first to show clean visualization of the visual reasoning process carried out by neural module networks, as opposed to gradient norms [113] or soft attention maps [97], [174].

In sum, this chapter makes the following contributions:

- We construct CLEVR-Ref+, a synthetic diagnostic dataset for referring expression tasks that complements existing real-world datasets.
- We test and diagnose several state-of-the-art referring expression models on CLEVR-

Ref+, including our proposed IEP-Ref that explicitly captures compositionality.

- The segmentation module trained in IEP-Ref can be trivially plugged in all intermediate steps in the module network to produce excellent segmentation masks that clearly reveal the network's reasoning process.

7.2 Related Works

7.2.1 Referring Expressions

Referring expressions are sentences that refer to specific objects in an image. Understanding referring expressions has important applications in robotics and human-computer interaction. In recent years, many deep learning models have been developed for this task.

Several works focused on detection, i.e. returning one bounding box containing the referred object. [102], [170] adapted image captioning for this task by scoring each bounding box proposal with a generative captioning model. [212] learned the alignment between the description and image region by reconstructing the description using an attention mechanism. [185], [283] studied the importance of context for referring expressions. [165] used a discriminative comprehension model to improve referring expression generation. [284] showed additional gain by incorporating reinforcement learning. [99], [282] used learned parser and module networks to better match the structured semantics.

There are also works focusing on segmentation, i.e. returning the segmentation mask. [100] used FCN feature concatenated with LSTM feature to produce pixel-wise binary segmentation. [152] used a convolutional LSTM in addition to the language-only LSTM to facilitate propagation of intermediate segmentation beliefs. [143], [173] improved upon [152] by making more architectural improvements.

7.2.2 Dataset Bias and Diagnostic Datasets

In visual question answering, despite exciting models being proposed and accuracy on benchmark datasets being steadily improved, there has been serious concern over the dataset bias problem [80], [288], meaning that models may be heavily exploiting the imbalanced distribution in the training/testing data. More recently, [41] showed that dataset bias also exists in referring expression datasets [123], [170], [283]. For example, [41] reported that the performance when discarding the referring expression and basing solely on the image is significantly higher than random. Ideally the dataset should be unbiased so that the performance faithfully reflect the model's true level of understanding. But this is very hard to control when working with real-world images and human-annotated referring expressions.

A possible solution is to use synthetic datasets. Indeed this is the path taken by CLEVR [112], a diagnostic dataset for VQA. There, objects are placed on a 2D plane and only have a small number of choices in terms of shape, color, size, and material. The question-answer pairs are also synthesized using carefully designed templates. Together with a uniform sampling strategy, this design can mitigate dataset bias and reveal the model's ability to understand compositionality. We construct our CLEVR-Ref+ dataset by re-purposing CLEVR towards the referring expression task.

Several approaches now achieve near-perfect accuracy on CLEVR [97], [98], [106], [113], [174], [201], [216]. In addition to reporting the VQA accuracy, they typically try to interpret the visual reasoning process through visualization. However, the quality of these visualizations does not match the high VQA accuracy. We suspect the primary reason is that the domain these models are trained for (i.e. a textual answer) is different from the domain these models are diagnosed on (i.e. attention over the image). Fortunately, in referring expressions these two domains are very much interchangeable.

Note that CLEVR was also adapted towards referring expression in [97], but they

focused on facilitating VQA, instead of introducing extensions (Section 7.3.3), evaluating state-of-the-art models (Section 7.4.1), and directly facilitating the diagnosis of visual reasoning (Section 7.4.3).

7.3 The CLEVR-Ref+ Dataset

CLEVR-Ref+ uses the exact same scenes as CLEVR (70K images in train set, 15K images in validation and test set), and every image is associated with 10 referring expressions. Since CLEVR is a VQA dataset, we began by changing the questions to referring expressions (Section 7.3.1), and the answers to referred objects (Section 7.3.2). We then made important additions to the set of modules (Section 7.3.3) as well as necessary changes to the sampling procedure (Section 7.3.4). Finally, we made the distinction whether more than one object is being referred (Section 7.3.5).

7.3.1 From Question to Referring Expression

Templates are provided in CLEVR so that questions and the functional programs associated with them can be generated at the same time. We notice that in many cases, part of the question is indeed a referring expression, as we need to first identify objects of interest before asking about their property (e.g. color or number). In Table 7.1 we provide examples of how we change question templates into their corresponding referring expression templates, usually by selecting a subset. The associated functional programs are also adjusted accordingly. For example, for "How many" questions, we simply remove the `Count` module at the end.

The original categories "Compare Integer" and "Comparison" were about comparing properties of two groups of referred objects, so they do not contribute additional referring expression patterns. Therefore they are not included in the templates for CLEVR-Ref+.

Table 7.1. Examples of converting questions to referring expressions.

Category	Question (CLEVR)	Referring Expression (CLEVR-Ref+)
Basic	How many cyan cubes are there?	The cyan cubes.
Spatial Relation	Are there any green cylinders to the left of the brown sphere?	The green cylinders to the left of the brown sphere.
AND Logic	How many green spheres are both in front of the red cylinder and left to the yellow cube?	The green spheres that are both in front of the red cylinder and left to the yellow cube.
OR Logic	Are there any cylinders that are either purple metal objects or small red matte things?	Cylinders that are either purple metal objects or small red matte things.
Same Relation	Are there any other things that have the same size as the red sphere?	The things/objects that have the same size as the red sphere.
Compare Integer	Are there more brown shiny objects behind the large rubber cylinder than gray blocks?	-
Comparison	Does the small ball have the same color as the small cylinder in front of the big sphere?	-

7.3.2 From Answer to Referred Objects

In referring expressions, the output is no longer a textual answer, but a bounding box or segmentation mask.

Since we know the exact 3D locations and properties of objects in the scene, we can follow the ground truth functional program associated with the referring expression to identify which objects are being referred. In fact we can do this not only at the end (also available in real-world datasets), but also at every intermediate step (not available in real-world datasets). This will become useful later when we do step-by-step inspection and evaluation of the visual reasoning process.

After finding the referred objects, we project them back to the image plane to get the ground truth bounding box and segmentation mask. This automatic annotation was done through rendering with the software Blender. For occluded objects, only the visible part is treated as ground truth.

7.3.3 Module Additions

We hope the referring expressions that we generate are representative of those used in the real world. However, since the task is no longer the same, we suspect that there may be some frequent referring patterns missing in the templates directly inherited from CLEVR. To this end, we analyzed statistics from a real-world referring expression dataset, RefCOCO+ [283], as shown in Table 7.2.

We began by sorting the words in these referring expressions by their frequency. Then, starting with the most frequent word, we empirically cluster these words into categories. Not surprisingly, nouns that represent object or human are the most common. However, going down the list, we found that the "ordinal" (e.g. "The second woman from left") and "visible" (e.g. "The barely seen backpack") categories recall more than 10% of all sentences, but are not included in the existing templates. Moreover, it is indeed possible to define

Table 7.2. Frequent category and words in RefCOCO+ [283].

Category	Example words	Frequency
object	shirt,head,chair,hat,pizza	63.66%
human	man,woman,guy,girl,person	42.54%
color	white,black,blue,red,green	38.76%
spatial	back,next,behind,near,up	23.86%
animal	zebra,elephant,horse,bear	15.36%
attribute	big,striped,small,plaid,long	10.55%
action	standing,holding,looking	10.34%
ordinal	**closest,furthest,first,third**	**5.797%**
compare	smaller,tallest,shorter,older	5.247%
visible	**fully visible,barely seen**	**4.639%**

them using a computer program, because there is no ambiguity in meaning. We add these two new modules into the CLEVR-Ref+ function catalog.

In a functional program, these two modules may be inserted whenever color, material, size, or shape is being described. As an example, "the red sphere" may be equivalently described as "the third sphere from left" or "the partially visible red object". In our dataset, we define an object to be *partially visible* if foreground objects' mask occupies more than 20% of its bounding box area. For an object to be *fully visible*, this value must be exactly 0. We do not describe visibility when there is an ambiguous case (i.e. this value is between 0 and 0.2) in the scene.

7.3.4 Generation Procedure

Generating a referring expression for a scene is conceptually simple and intuitive. The process may be summarized as the following few steps:

1. Randomly choose a referring expression family[1].
2. Randomly choose a text template from this family.

[1] A referring expression family contains a template for constructing functional programs and several text templates that provide multiple ways of expressing these programs in natural language.

3. Follow the functional program and select random values when encountering template parameters[2].
4. Reject when certain criteria fail, that is, the sampled referring expression is inappropriate for the given scene; return when the entire functional program follows through.

We largely follow the generation procedure of CLEVR, with a few important changes:

- To balance the number of referring expressions across different categories (those listed in Table 7.1), we double the probability of being sampled in categories with a small number of referring expression families.
- When describing the attributes for a set of objects, we do not use `Ordinal` and `Visible` at the same time. This is because referring an object as "The second partially visible object from left" seems too peculiar and rare, and there usually exists more natural alternatives.
- Originally when describing the attributes for a set of objects, four fair coins were flipped to determine whether color, material, size, shape will be included. As a result, usually multiple attributes are selected, and a very small number of objects survive these filters. We empirically found that this makes it quite easy for the system to select the correct object simply from the attributes that directly describe the target object(s).

  To remedy this, we first enumerate all possible combinations of these attributes, and calculate how many objects will survive for each possibility. We then uniformly sample from these possible number of survivors, before doing another uniform sampling to find the combination of attributes. This will ensure a larger variance in terms of number of objects after each set of filtering, and prevent near-degenerate solutions.

[2] For instance, left/right/front/behind; big/small; metal/rubber.

- At the end of the functional program, we verify if at least one object is being referred; reject otherwise.

7.3.5 Multi-Object and Single-Object Referring

As explained in Section 7.3.4, each referring expression in CLEVR-Ref+ may refer to one or more objects in the scene. We believe this is the more general setting, and models should have the flexibility to handle various number of objects being referred. This is already handled and supported by referring image segmentation systems. However, we notice that detection based systems are usually designed to return a single object instead of multiple objects, presumably because this was how the detection datasets [170], [283] were created. As a result, for detection based methods, we evaluate on the subset of CLEVR-Ref+ where only a single object is referred. This subset contains a total of 222,569 referring expressions (32% of the entire dataset).

7.4 Experiments

7.4.1 Models and Implementation Details

In all models we resize the input image to 320×320 to set up a fair comparison. Publicly available code for these models are used with minimum change to adapt to our CLEVR-Ref+ dataset. The following referring expression models are studied and tested:

Speaker-Listener-Reinforcer (SLR) [284] This is a **detection** model that includes a generative model (speaker), a discriminative model (listener), as well as a reinforcement learning component that makes further improvement. Before training the main model, the visual-language similarity model needs to be trained first. We use Adam optimizer [125], learning rate 4e-4, batch size 32 for both the visual-language similarity model and the main model.

MAttNet [282] This is also a **detection** model, that uses three modular networks to capture the subject, location, and relationship features respectively. A soft attention mechanism is used to return the overall score of a candidate region. We use learning rate 4e-4 and batch size 15.

Recurrent Multimodal Interaction (RMI) [152] This is a **segmentation** model. In addition to concatenating the referring expression LSTM embedding with the image features, RMI also used a convolutional LSTM to facilitate propagation of segmentation beliefs when reading in the referring expression word-by-word. We use Adam optimizer, learning rate 2.5e-4, batch size 3, and weight decay 5e-4.

IEP-Ref This is a **segmentation** model that we adapt from IEP [113], which was designed for VQA. The idea is to use a LSTM program generator to translate the referring expression into a structured series of modules, each of which is parameterized by a small CNN. By executing this dynamically constructed neural network (with a special `Segment` module at the end; see Appendix C for its architecture), IEP-Ref imitates the underlying visual reasoning process. For input visual features, we use the last layer of the `conv4` stage of ResNet101 [89] pretrained on ImageNet [47], which is of size $1024 \times 20 \times 20$. Following [113], this part is not finetuned. We tried three settings that use 9K/18K/700K ground truth programs to train the LSTM program generator (Adam optimizer, learning rate 5e-4, batch size 64; 20,000 iterations for the 9K setting, 32,000 iterations for the 18K and 700K setting). The accuracies of the predicted programs are 0.873, 0.971, 0.993 respectively. For the fourth setting, we simply use the ground truth program[3]. The execution engine is trained for 30 epochs using learning rate 1e-4 and Adam optimizer.

[3] This is our default IEP-Ref setting unless otherwise specified.

7.4.2 Results and Analysis

7.4.2.1 Overall Evaluation

The experimental results are summarized in Table 7.3. Detection models are evaluated by accuracy (i.e. whether the prediction selects the correct bounding box among given candidates), where MAttNet performs favorably against SLR. Segmentation models are evaluated by Intersection over Union (IoU), where IEP-Ref performs significantly better than RMI. This suggests the importance to model compositionality within the referring expression. We now present a more detailed analysis of various aspects.

7.4.2.2 Basic Referring Ability

Here we start with the easiest form: referring by direct description of object attributes (e.g., "The big blue sphere"). Concretely, this corresponds to the "0-Relate" subset.

In CLEVR-Ref+, there are totally 6 types of attributes that may help us locate specific objects: color, size, shape, material, ordinality, and visibility. In Figure 7.2 we show the average detection accuracy/segmentation IoU of various methods on "0-Relate" referring expressions that either contain or not contain a specific type of module.

Among detection models, we found that accuracy is higher when the referring expression contains descriptions of color, shape, and visibility. A reasonable conjecture is that these concepts are easier to learn compared with the others. However, for segmentation, the performance gaps between "exclude" and "include" are not as significant.

Though it is unclear which concept is the easiest to learn, there seems little dispute that ordinality is the hardest. In particular, for RMI, IoU is 0.91 if the expression does not require ordinality and 0.27 when it does. Other models do not suffer as much, but also experience significant drops. We suspect this is because ordinality requires the global context, whereas the others are local properties.

Table 7.3. Referring object detection and referring image segmentation results on CLEVR-Ref+. We evaluated three existing models, as well as IEP-Ref which we adapted from its VQA counterpart.

	Basic	Spatial Relation			Logic				
	0-Relate	1-Relate	2-Relate	3-Relate	AND	OR	Same	Accuracy	IoU
SLR [284]	0.627	0.569	0.570	0.584	0.594	0.701	0.444	0.577	-
MAttNet [282]	0.566	0.623	0.634	0.624	0.723	0.737	0.454	0.609	-
RMI [152]	0.822	0.713	0.736	0.715	0.585	0.679	0.251	-	0.561
IEP-Ref (GT)	0.928	0.895	0.908	0.908	0.879	0.881	0.647	-	0.816
IEP-Ref (700K prog.)	0.920	0.884	0.902	0.898	0.860	0.869	0.636	-	0.806
IEP-Ref (18K prog.)	0.907	0.858	0.874	0.862	0.829	0.847	0.605	-	0.782
IEP-Ref (9K prog.)	0.910	0.858	0.847	0.811	0.778	0.791	0.626	-	0.760

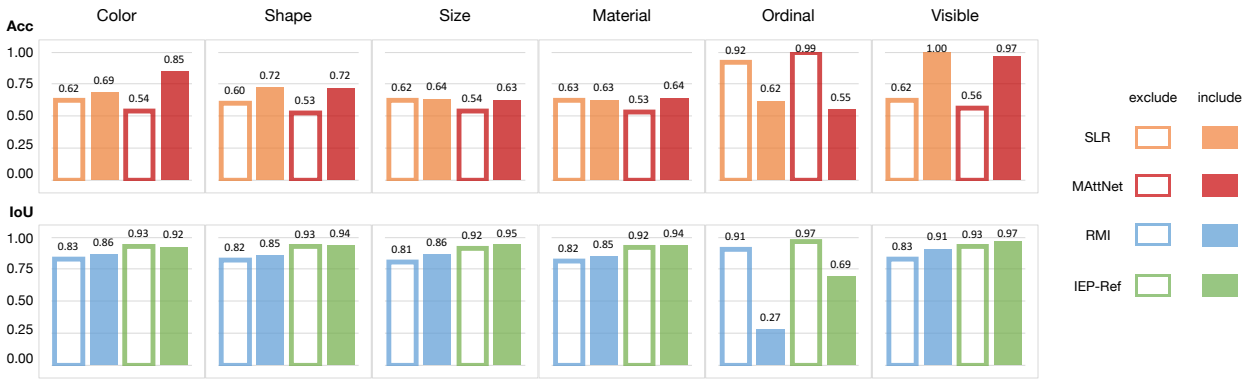


Figure 7.2. Analyzing the basic referring ability of different models. "Include" means the average performance if a module is involved in the referring process. "Exclude" means otherwise. As a result, high "exclude" and low "include" performance suggests that this module is more challenging to learn, and vice versa.

7.4.2.3 Spatial Reasoning Ability

Other than directly describing the attributes, it is also common to refer to an object by its spatial location. Here we diagnose whether referring expression models can understand (potentially multiple steps of) relative spatial relationship, for example "The object that is left to the red cube". In Table 7.3, this corresponds to the "{0, 1, 2, 3}-Relate" columns. Results are shown in Figure 7.3.

In general, we observe a small drop when referring expressions start to include spatial reasoning. However, there does not seem to be significant difference among referring expressions that require 1, 2, 3 steps of spatial reasoning. This seems to suggest that once the model has grasped spatial reasoning, there is little trouble in successfully applying it multiple times.

7.4.2.4 Different Reasoning Topologies

There are two referring expression topologies in CLEVR-Ref+: chain-structured and tree-structured. Intuitively, a chain structure has a single reasoning path to follow, whereas a tree structure requires following two such paths before merging. In Figure 7.4 we compare

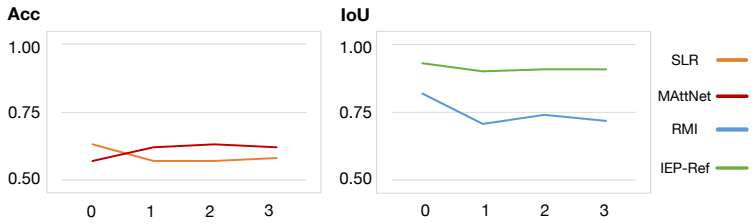


Figure 7.3. Analyzing the spatial reasoning ability of different models. Horizontal axis is the number of spatial relations.

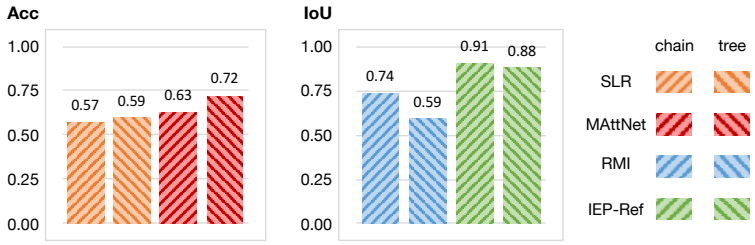


Figure 7.4. Effect of reasoning topology (Chain vs. Tree) on referring detection or segmentation performance.

performance on referring expressions with two sequential spatial relationships vs. one on each branch joined with AND. These two templates have roughly the same length and complexity, so the comparison focuses on topology.

Though not consistent among the four models, tree-structured referring expressions are generally harder than chain-structured ones. This agrees with the findings in [112].

7.4.2.5 Different Relation Types

There are two kinds of relationships in CLEVR-Ref+. One is spatial relationship that includes phrases like "left of", "right of", "in front of", "behind" (discussed in Section 7.4.2.3). The other is same-attribute relationship that requires recognizing and memorizing particular attributes of another object, e.g. "The large block(s) that have the same color as the metal sphere".

In Figure 7.5 we study whether the relation type will make a difference in performance.

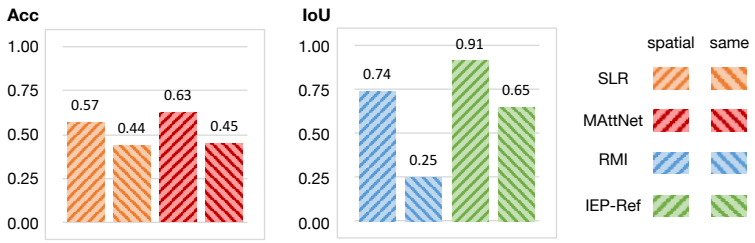


Figure 7.5. Effect of relation type (Spatial vs. Same) on referring detection or segmentation performance.

We compare the "2-Relate" column with the "Same" column in Table 7.3, again because they have roughly the same length and complexity. All models perform much worse on the same-attribute relationship type, suggesting that this is a hard concept to grasp. Similar to ordinality, same-attribute requires global context.

7.4.3 Step-By-Step Inspection of Visual Reasoning

All the results discussed in Section 7.4.2 are about the endpoint of the visual reasoning process. We argue that in order to trust the predictions made by the referring expression system, it is also important to make sure that the intermediate reasoning steps make sense. CLEVR-Ref+ is suitable because: (1) the semantics of the referring expressions is modularized, and (2) the referring ground truth at all intermediate steps can be obtained automatically (i.e. no human annotators needed).

In training our IEP-Ref model, there is always a `Segment` module at the end, transforming the 128-channel feature map into a 1-channel segmentation mask. When testing, we simply attach the trained `Segment` module to the output of all intermediate modules. This is possible because all modules have the same number of input channels and output channels (128). This technique would not help in the VQA setting, because there the ending modules (e.g. `Count`, `Equal`) discard the spatial dimensions needed for visualization.

We found that this technique works quite well. In Figure 7.6 we provide four qualitative examples with various topologies and modules. We notice that all modules are performing

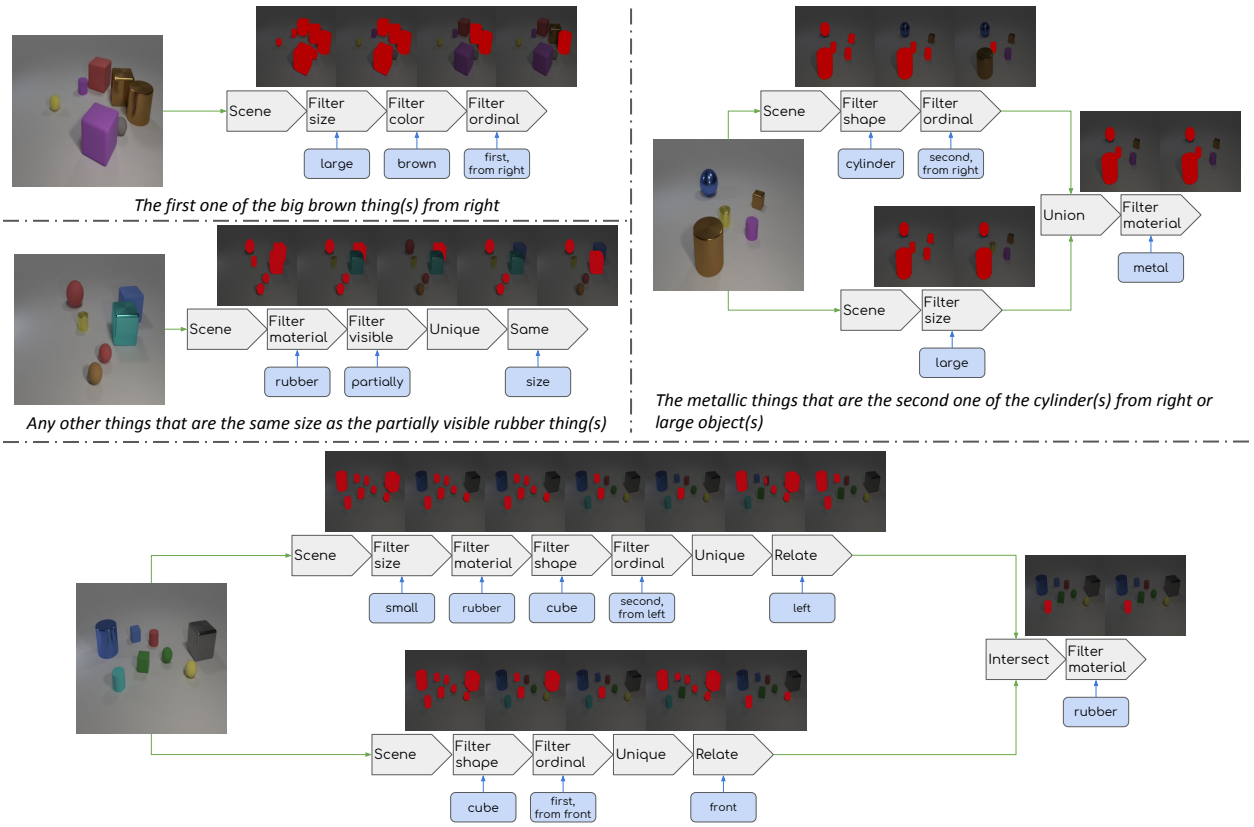


Figure 7.6. Four examples (two chain structures, two tree structures) of step-by-step inspection of IEP-Ref visual reasoning.

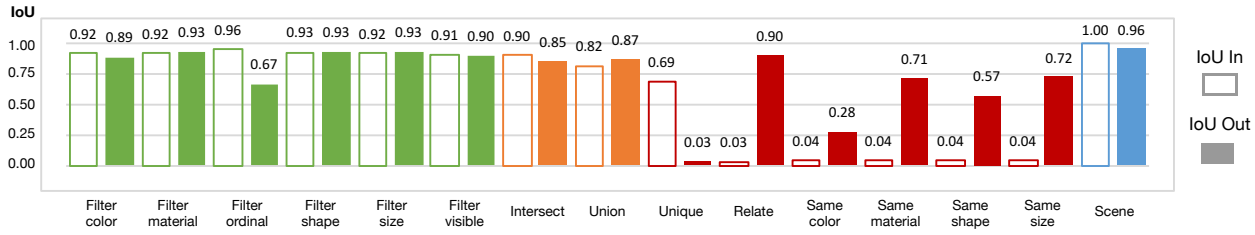


Figure 7.7. Average IoU going into/out of each IEP-Ref module on CLEVR-Ref+ validation set. Note that here IoU is not only computed at the end, but also all intermediate steps. This shows that IoU remains high throughout visual reasoning. The large differences in modules marked in dark red are discussed in text.

their intended functionality, except the `Unique` module[4]. Yet after one more module,

[4] It is supposed to simply carry over the previously referred object, yet from what we observe, its behavior is most similar to selecting the complement of the previously referred object, though this is far from consistent.

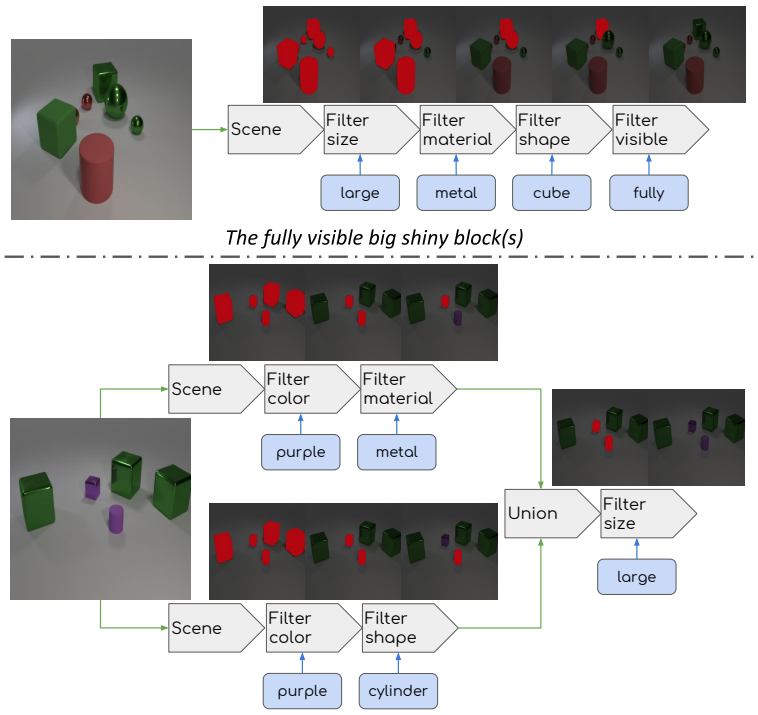


Figure 7.8. Our IEP-Ref model can correctly handle false-premise referring expressions even if they do not appear during training.

the segmentation mask becomes normal again. The quantitative analysis in Figure 7.7 confirms this observation: on average, IoU drops by 0.66 after each `Unique` module; but IoU significantly increases after each `Same` or `Relate` module, and these are the only modules that may come after `Unique` according to the templates. We conjecture that the network has learned some mechanism to treat `Unique` as the "preprocessing" step of the `Same` and `Relate` functionalities.

7.4.4 False-Premise Referring Expressions

In reality, referring expression systems may face all kinds of textual input, and not all of them will make sense. When presented with a referring expression that makes false assumptions (e.g. "The red sphere" when there is no sphere in the scene), the system should follow through as much as it can, and be robust enough to return zero foreground

at the end. We test IEP-Ref's ability to deal with these false-premise referring expressions (c.f. [207]). Note that no such expressions appear during training.

We generate 10,000 referring expressions that refer to zero object at the end. Qualitatively (see Figure 7.8), it is reassuring to see that intermediate modules are correctly doing their jobs, and a no-foreground prediction is made at the final step. Quantitatively, IEP-Ref predicts 0 foreground pixel more than $1/4$ of the time, and ≤ 8 foreground pixels more than $1/3$ of the time.

7.5 Conclusion

In this chapter, we build the CLEVR-Ref+ dataset to complement existing ones for referring expressions. By choosing a synthetic setup, the advantage is that dataset bias can be minimized, and the ground truth visual reasoning process is readily available. We evaluated several state-of-the-art referring object detection and referring image segmentation models on CLEVR-Ref+. In addition, we propose the IEP-Ref model, which uses a module network approach and outperforms competing methods by a large margin. Detailed analysis are conducted to identify the strengths and weaknesses of these models. In particular, we found that ordinality and the same-attribute relationship seem to be the most difficult concepts to grasp.

Besides the correctness of the final segmentation mask, the correctness of the reasoning process is also important. We discovered that IEP-Ref provides an easy and natural way of revealing this process: simply attach the `Segment` module to each intermediate step. Our quantitative evaluation shows a high IoU at intermediate steps as well, proving that the neural modules have indeed learned the job they are supposed to do. Another evidence is that IEP-Ref can correctly handle false-premise referring expressions.

Going forward, we are interested to see whether these findings will transfer and inspire better models on real data.

Chapter 8

Scene Graph Parsing as Dependency Parsing

This chapter presents a streamlined method for parsing a sentence into a scene graph, a representation that is easily interpretable and could benefit vision applications.

8.1 Introduction

Recent years have witnessed the rise of interest in many tasks at the intersection of computer vision and natural language processing, including semantic image retrieval [114], [252], image captioning [53], [120], [153], [171], visual question answering [4], [5], [295], and referring expressions [102], [152], [170]. The pursuit for these tasks is in line with people's desire for high level understanding of visual content, in particular, using textual descriptions or questions to help understand or express images and scenes.

What is shared among all these tasks is the need for a *common representation* to establish connection between the two different modalities. The majority of recent works handle the vision side with convolutional neural networks, and the language side with recurrent neural networks [38], [92] or word embeddings [179], [200]. In either case, neural networks map original sources into a semantically meaningful [52], [179] vector representation that can be aligned through end-to-end training [68]. This suggests that the vector embedding space is an appropriate choice as the common representation connecting different

Figure 8.1. Each image in the Visual Genome [131] dataset contains tens of region descriptions and the region scene graphs associated with them. In this chapter, we study how to generate high quality scene graphs (two such examples are shown in the figure) from textual descriptions, without using image information.

modalities (see e.g. [117]).

While the dense vector representation yields impressive performance, it has an unfortunate limitation of being less intuitive and hard to interpret. Scene graphs [114], on the other hand, proposed a type of directed graph to encode information in terms of objects, attributes of objects, and relationships between objects (see Figure 8.1 for visualization). This is a more structured and explainable way of expressing the knowledge from either modality, and is able to serve as an alternative form of common representation. In fact, the value of scene graph representation has already been proven in a wide range of visual tasks, including semantic image retrieval [114], caption quality evaluation [3], etc. In this chapter, we focus on scene graph generation from textual descriptions.

Previous attempts at this problem [3], [219] follow the same spirit. They first use a dependency parser to obtain the dependency relationship for all words in a sentence, and then use either a rule-based or a learned classifier as post-processing to generate the scene graph. However, the rule-based classifier cannot learn from data, and the learned classifier

is rather simple with hand-engineered features. In addition, the dependency parser was trained on linguistics data to produce complete dependency trees, some parts of which may be redundant and hence confuse the scene graph generation process.

Therefore, our model abandons the two-stage pipeline, and uses a single, customized dependency parser instead. The customization is necessary for two reasons. First is the difference in label space. Standard dependency parsing has tens of edge labels to represent rich relationships between words in a sentence, but in scene graphs we are only interested in three types, namely objects, attributes, and relations. Second is whether every word needs a head. In some sense, the scene graph represents the "skeleton" of the sentence, which suggests that empty words are unlikely to be included in the scene graph. We argue that in scene graph generation, it is unnecessary to require a parent word for every single word.

We build our model on top of a neural dependency parser implementation [126] that is among the state-of-the-art. We show that our carefully customized dependency parser is able to generate high quality scene graphs by learning from data. Specifically, we use the Visual Genome dataset [131], which provides rich amounts of region description - region graph pairs. We first align nodes in region graphs with words in the region descriptions using simple rules, and then use this alignment to train our customized dependency parser. We evaluate our parser by computing the F-score between the parsed scene graphs and ground truth scene graphs. We also apply our approach to image retrieval to show its effectiveness.

8.2 Related Works

8.2.1 Scene Graphs

The scene graph representation was proposed in [114] as a way to represent the rich, structured knowledge within an image. The nodes in a scene graph represent either an

object, an attribute for an object, or a relationship between two objects. The edges depict the connection and association between two nodes. This representation is later adopted in the Visual Genome dataset [131], where a large number of scene graphs are annotated through crowd-sourcing.

The scene graph representation has been proved useful in various problems including semantic image retrieval [114], visual question answering [248], 3D scene synthesis [25], and visual relationship detection [163]. Excluding [114] which used ground truth, scene graphs are obtained either from images [45], [144], [270] or from textual descriptions [3], [219]. In this chapter we focus on the latter.

In particular, parsed scene graphs are used in [219] for image retrieval. We show that with our more accurate scene graph parser, performance on this task can be further improved.

8.2.2 Parsing to Graph Representations

The goal of dependency parsing [135] is to assign a parent word to every word in a sentence, and every such connection is associated with a label. Dependency parsing is particularly suitable for scene graph generation because it directly models the relationship between individual words without introducing extra nonterminals. In fact, all previous work [3], [219] on scene graph generation run dependency parsing on the textual description as a first step, followed by either heuristic rules or simple classifiers. Instead of running two separate stages, our work proposed to use a single dependency parser that is end-to-end. In other words, our customized dependency parser generates the scene graph in an online fashion as it reads the textual description once from left to right.

In recent years, dependency parsing with neural network features [26], [44], [56], [57], [126], [224] has shown impressive performance. In particular, [126] used bidirectional LSTMs to generate features for individual words, which are then used to predict parsing actions. We base our model on [126] for both its simplicity and good performance.

Apart from dependency parsing, Abstract Meaning Representation (AMR) parsing [66], [128], [254], [259] may also benefit scene graph generation. However, as first pointed out in [3], the use of dependency trees still appears to be a common theme in the literature, and we leave the exploration of AMR parsing for scene graph generation as future work.

More broadly, our task also relates to entity and relation extraction, e.g. [121], but there object attributes are not handled. Neural module networks [4] also use dependency parses, but they translate questions into a series of actions, whereas we parse descriptions into their graph form. Finally, [132] connected parsing and grounding by training the parser in a weakly supervised fashion.

8.3 Task Description

In this section, we begin by reviewing the scene graph representation, and show how its nodes and edges relate to the words and arcs in dependency parsing. We then describe simple yet reliable rules to align nodes in scene graphs with words in textual descriptions, such that customized dependency parsing, described in the next section, may be trained and applied.

8.3.1 Scene Graph Definition

There are three types of nodes in a scene graph: object, attribute, and relation. Let $\mathcal{O}$ be the set of object classes, $\mathcal{A}$ be the set of attribute types, and $\mathcal{R}$ be the set of relation types. Given a sentence s, our goal in this chapter is to parse s into a scene graph:

$$G(s) = \langle O(s), A(s), R(s) \rangle \tag{8.1}$$

where $O(s) = \{o_1(s), \ldots, o_m(s)\}, o_i(s) \in \mathcal{O}$ is the set of object instances mentioned in s, $A(s) \subseteq O(s) \times \mathcal{A}$ is the set of attributes associated with object instances, and $R(s) \subseteq O(s) \times \mathcal{R} \times O(s)$ is the set of relations between object instances.

$G(s)$ is a graph because we can first create an object node for every element in $O(s)$; then for every (o, a) pair in $A(s)$, we create an attribute node and add an unlabeled edge $o \rightarrow a$; finally for every (o_1, r, o_2) triplet in $R(s)$, we create a relation node and add two unlabeled edges $o_1 \rightarrow r$ and $r \rightarrow o_2$. The resulting directed graph exactly encodes information in $G(s)$. We call this the **node-centric** graph representation of a scene graph.

We realize that a scene graph can be equivalently represented by no longer distinguishing between the three types of nodes, yet assigning labels to the edges instead. Concretely, this means there is now only one type of node, but we assign a `ATTR` label for every $o \rightarrow a$ edge, a `SUBJ` label for every $o_1 \rightarrow r$ edge, and a `OBJT` label for every $r \rightarrow o_2$ edge. We call this the **edge-centric** graph representation of a scene graph.

We can now establish a connection between scene graphs and dependency trees. Here we only consider scene graphs that are acyclic[1]. The edge-centric view of a scene graph is very similar to a dependency tree: they are both directed acyclic graphs where the edges/arcs have labels. The difference is that in a scene graph, the nodes are the objects/attributes/relations and the edges have label space {`ATTR`, `SUBJ`, `OBJT`}, whereas in a dependency tree, the nodes are individual words in a sentence and the edges have a much larger label space.

8.3.2 Sentence-Graph Alignment

We have shown the connection between nodes in scene graphs and words in dependency parsing. With alignment between nodes in scene graphs and words in the textual description, scene graph generation and dependency parsing becomes equivalent: we can construct the generated scene graph from the set of labeled edges returned by the dependency parser. Unfortunately, such alignment is not provided between the region graphs and region descriptions in the Visual Genome [131] dataset. Here we describe how we use simple yet reliable rules to do sentence-graph (word-node) alignment.

[1] In Visual Genome, only 4.8% region graphs have cyclic structures.

Algorithm 8.1: First cycle of the alignment procedure.

```
1  Input: Sentence s; Scene graph G(s)
2  Initialize aligned nodes N as empty set
3  Initialize aligned words W as empty set
4  for o in object nodes of G(s) \ N do
5      for w in s \ W do
6          if o ↔ w according to WBW then
7              Add (o, w); N = N ∪ {o}; W = W ∪ {w}
8  for a in attribute nodes of G(s) \ N do
9      for w in s \ W do
10         if a ↔ w according to WBW or SYN and a's object node is in N then
11             Add (a, w); N = N ∪ {a}; W = W ∪ {w}
12 for r in relation nodes of G(s) \ N do
13     for w in s \ W do
14         if r ↔ w according to WBW or SYN and r's subject and object nodes are both
           in N then
15             Add (r, w); N = N ∪ {r}; W = W ∪ {w}
```

There are two strategies that we could use in deciding whether to align a scene graph node d (whose label space is $\mathcal{O} \cup \mathcal{A} \cup \mathcal{R}$) with a word/phrase w in the sentence:

- Word-by-word match (WBW): $d \leftrightarrow w$ only when d's label and w match word-for-word.
- Synonym match (SYN)[2]: $d \leftrightarrow w$ when the wordnet synonyms of d's label contain w.

Obviously WBW is a more conservative strategy than SYN.

We propose to use two cycles and each cycle further consists of three steps, where we try to align objects, attributes, relations in that order. The pseudocode for the first cycle is in Algorithm 8.1. The second cycle repeats line 4-15 immediately afterwards, except that in line 6 we also allow SYN. Intuitively, in the first cycle we use a conservative strategy to find "safe" objects, and then scan for their attributes and relations. In the second cycle

[2] This strategy is also used in [48] and [3].

we relax and allow synonyms in aligning object nodes, also followed by the alignment of attribute and relation nodes.

The ablation study of the alignment procedure is reported in the experimental section.

8.4 Customized Dependency Parsing

In the previous section, we have established the connection between scene graph generation and dependency parsing, which assigns a parent word for every word in a sentence, as well as a label for this directed arc. We start by describing our base dependency parsing model, which is neural network based and performs among the state-of-the-art. We then show why and how we do customization, such that scene graph generation is achieved with a single, end-to-end model.

8.4.1 Neural Dependency Parsing Base Model

We base our model on the transition-based parser of [126]. Here we describe its key components: the arc-hybrid system that defines the transition actions, the neural architecture for feature extractor and scoring function, and the loss function.

The Arc-Hybrid System In the arc-hybrid system, a configuration consists of a stack σ, a buffer β, and a set T of dependency arcs. Given a sentence $s = w_1, \ldots, w_n$, the system is initialized with an empty stack σ, an empty arc set T, and $\beta = 1, \ldots, n, \texttt{ROOT}$, where ROOT is a special index. The system terminates when σ is empty and β contains only ROOT. The dependency tree is given by the arc set T upon termination.

The arc-hybrid system allows three transition actions, SHIFT, LEFT$_l$, RIGHT$_l$, described in Table 8.1. The SHIFT transition moves the first element of the buffer to the stack. The LEFT(l) transition yields an arc from the first element of the buffer to the top element of the stack, and then removes the top element from the stack. The RIGHT(l) transition yields

Stack σ_t	**Buffer** β_t	**Arc set** T_t	**Action**	**Stack** σ_{t+1}	**Buffer** β_{t+1}	**Arc set** T_{t+1}
σ	$b_0\|\beta$	T	SHIFT	$\sigma\|b_0$	β	T
$\sigma\|s_1\|s_0$	$b_0\|\beta$	T	LEFT(l)	$\sigma\|s_1$	$b_0\|\beta$	$T \cup \{(b_0, s_0, l)\}$
$\sigma\|s_1\|s_0$	β	T	RIGHT(l)	$\sigma\|s_1$	β	$T \cup \{(s_1, s_0, l)\}$
$\sigma\|s_0$	β	T	REDUCE	σ	β	T

Table 8.1. Transition actions under the arc-hybrid system. The first three actions are from dependency parsing; the last one is introduced for scene graph parsing.

an arc from the second top element of the stack to the top element of the stack, and then also removes the top element from the stack.

The following paragraphs describe how to select the correct transition action (and label l) in each step in order to generate a correct dependency tree.

BiLSTM Feature Extractor Let the word embeddings of a sentence s be $\mathbf{w}_1, \ldots, \mathbf{w}_n$. An LSTM cell is a parameterized function that takes as input $\mathbf{w}_t$, and updates its hidden states:

$$\text{LSTM cell} : (\mathbf{w}_t, \mathbf{h}_{t-1}) \rightarrow \mathbf{h}_t \tag{8.2}$$

As a result, an LSTM network, which simply applies the LSTM cell t times, is a parameterized function mapping a sequence of input vectors $\mathbf{w}_{1:t}$ to a sequence of output vectors $\mathbf{h}_{1:t}$. In our notation, we drop the intermediate vectors $\mathbf{h}_{1:t-1}$ and let $\text{LSTM}(\mathbf{w}_{1:t})$ represent $\mathbf{h}_t$.

A bidirectional LSTM, or BiLSTM for short, consists of two LSTMs: LSTM_F which reads the input sequence in the original order, and LSTM_B which reads it in reverse. Then

$$\begin{aligned} &\text{BILSTM}(\mathbf{w}_{1:n}, i) = \\ &\qquad \text{LSTM}_F(\mathbf{w}_{1:i}) \circ \text{LSTM}_B(\mathbf{w}_{n:i}) \end{aligned} \tag{8.3}$$

where $\circ$ denotes concatenation. Intuitively, the forward LSTM encodes information from the left side of the i-th word and the backward LSTM encodes information to its right, such that the vector $\mathbf{v}_i = \text{BILSTM}(\mathbf{w}_{1:n}, i)$ has the full sentence as context.

When predicting the transition action, the feature function $\phi(c)$ that summarizes the current configuration $c = (\sigma, \beta, T)$ is simply the concatenated BiLSTM vectors of the top three elements in the stack and the first element in the buffer:

$$\phi(c) = \mathbf{v}_{s_2} \circ \mathbf{v}_{s_1} \circ \mathbf{v}_{s_0} \circ \mathbf{v}_{b_0} \tag{8.4}$$

MLP Scoring Function The score of transition action y under the current configuration c is determined by a multi-layer perceptron with one hidden layer:

$$f(c, y) = MLP(\phi(c))[y] \tag{8.5}$$

where

$$MLP(x) = W_2 \cdot \tanh(W_1 \cdot x + b_1) + b_2 \tag{8.6}$$

Hinge Loss Function The training objective is to raise the scores of correct transitions above scores of incorrect ones. Therefore, at each step, we use a hinge loss defined as:

$$\begin{aligned} \mathcal{L} = \max(0, 1 - & \max_{y^+ \in Y^+} f(c, y^+) \\ & + \max_{y^- \in Y \setminus Y^+} f(c, y^-)) \end{aligned} \tag{8.7}$$

where Y is the set of possible transitions and Y^+ is the set of correct transitions at the current step. In each training step, the parser scores all possible transitions using Equation 8.5, incurs a loss using Equation 8.7, selects a following transition, and updates the configuration. Losses at individual steps are summed throughout the parsing of a sentence, and then parameters are updated using backpropagation.

In test time, we simply choose the transition action that yields the highest score at each step.

8.4.2 Customization

In order to generate scene graphs with dependency parsing, modification is necessary for at least two reasons. First, we need to redefine the label space of arcs so as to reflect the edge-centric representation of a scene graph. Second, not every word in the sentence will be (part of) a node in the scene graph (see Figure 8.2 for an example). In other words, some words in the sentence may not have a parent word, which violates the dependency parsing setting. We tackle these two challenges by redesigning the edge labels and expanding the set of transition actions.

Redesigning Edge Labels We define a total of five edge labels, so as to faithfully bridge the edge-centric view of scene graphs with dependency parsing models:

- `CONT`: This label is created for nodes whose label is a phrase. For example, the phrase "in front of" is a single relation node in the scene graph. By introducing the `CONT` label, we expect the parsing result to be either

$$\text{in} \xrightarrow{\texttt{CONT}} \text{front} \xrightarrow{\texttt{CONT}} \text{of} \tag{8.8}$$

  or

$$\text{in} \xleftarrow{\texttt{CONT}} \text{front} \xleftarrow{\texttt{CONT}} \text{of} \tag{8.9}$$

  where the direction of the arcs (left or right) is predefined by hand.

  The leftmost word under the right arc rule or the rightmost word under the left arc rule is called the *head* of the phrase. A single-word node does not need this `CONT` label, and the head is itself.

- `ATTR`: The arc label from the head of an object node to the head of an attribute node.

- `SUBJ`: The arc label from the head of an object node (subject) to the head of a relation node.

- OBJT: The arc label from the head of a relation node to the head of an object node (object).
- BEGN: The arc label from the ROOT index to all heads of object nodes without a parent.

Expanding Transition Actions With the three transition actions SHIFT, LEFT(l), RIGHT(l), we only drop an element (from the top of the stack) after it has already been associated with an arc. This design ensures that an arc is associated with every word. However, in our setting for scene graph generation, there may be no arc for some of the words, especially empty words.

Our solution is to augment the action set with a REDUCE action, that pops the stack *without* adding to the arc set (see Table 8.1). This action is often used in other transition-based dependency parsing systems (e.g. arc-eager [192]). More recently, [90] and [20] also included this action when parsing sentences to graph structures.

We still minimize the loss function defined in Equation 8.7, except that now $|Y|$ increases from 3 to 4. During training, we impose the oracle to select the REDUCE action when it is in Y^+. In terms of loss function, we increment by 1 the loss incurred by the other 3 transition actions if REDUCE incurs zero loss.

8.5 Experiments

8.5.1 Implementation Details

We train and evaluate our scene graph parsing model on (a subset of) the Visual Genome [131] dataset. Each image in Visual Genome contains a number of regions, and each region is annotated with both a region description and a region scene graph. Our training set is the intersection of Visual Genome and MS COCO [148] train2014 set, which contains a total of 34027 images/ 1070145 regions. We evaluate on the intersection of Visual Genome and MS COCO val2014 set, which contains a total of 17471 images/ 547795 regions.

Parser	F-score
Stanford [219]	0.3549
SPICE [3]	0.4469
Ours (left arc rule)	**0.4967**
Ours (right arc rule)	0.4952
Ours (all SYN)	0.4877
Ours (no SYN)	0.4538
Oracle	0.6985

Table 8.2. The F-scores (i.e. SPICE metric) between scene graphs parsed from region descriptions and ground truth region graphs on the intersection of Visual Genome [131] and MS COCO [148] validation set.

In our experiments, the number of hidden units in BiLSTM is 256; the number of layers in BiLSTM is 2; the word embedding dimension is 200; the number of hidden units in MLP is 100. We use fixed learning rate 0.001 and Adam optimizer [125] with epsilon 0.01. Training usually converges within 4 epochs.

8.5.2 Quality of Parsed Scene Graphs

We use a slightly modified version of SPICE score [3] to evaluate the quality of scene graph parsing. Specifically, for every region, we parse its description using a parser (e.g. the one used in SPICE or our customized dependency parser), and then calculate the F-score between the parsed graph and the ground truth region graph (see Section 3.2 of [3] for more details). Note that when SPICE calculates the F-score, a node in one graph could be matched to several nodes in the other, which is problematic. We fix this and enforce one-to-one matching when calculating the F-score. Finally, we report the average F-score across all regions.

Table 8.2 summarizes our results. We see that our customized dependency parsing model achieves an average F-score of 49.67%, which significantly outperforms the parser used in SPICE by 5 percent. This result shows that our customized dependency parser is

very effective at learning from data, and generates more accurate scene graphs than the best previous approach.

Ablation Studies First, we study how the sentence-graph alignment procedure affects the final performance. Recall that our procedure involves two cycles, each with three steps. Of the six steps, synonym match (SYN) is only not used in the first step. We tried two more settings, where SYN is either used in all six steps or none of the six steps. We can see from Table 8.2 that the final F-score drops in both cases, hence supporting the procedure that we chose.

Second, we study whether changing the direction of `CONT` arcs from pointing left to pointing right will make much difference. Table 8.2 shows that the two choices give very similar performance, suggesting that our dependency parser is robust to this design choice.

Finally, we report the oracle score, which is the similarity between the aligned graphs that we use during training and the ground truth graphs. The F-score is relatively high at 69.85%. This shows that improving the parser (about 20% margin) and improving the sentence-graph alignment (about 30% margin) are both promising directions for future research.

Qualitative Examples We provide one parsing example in Figure 8.2 and Figure 8.3. This is a sentence that is relatively simple, and the underlying scene graph includes two object nodes, one attribute node, and one compound word relation node. In parsing this sentence, all four actions listed in Table 8.1 are used (see Figure 8.3) to produce the edge-centric scene graph (bottom left of Figure 8.2), which is then trivially converted to the node-centric scene graph (bottom right of Figure 8.2).

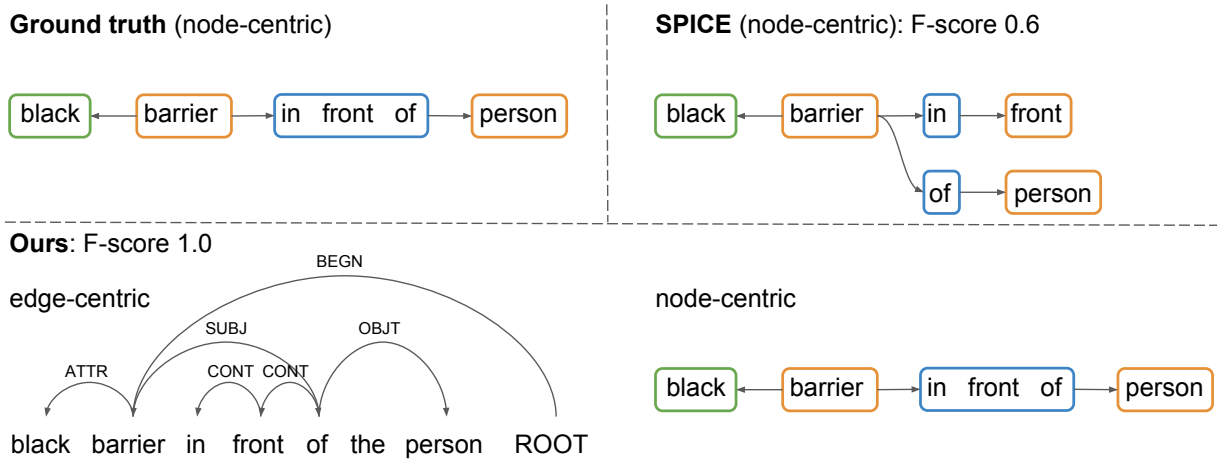


Figure 8.2. Scene graph parsing result of the sentence "black barrier in front of the person". In the node-centric graphs, orange represents object node, green represents attribute node, blue represents relation node.

	Stack	Buffer	Action
0		black barrier in front of the person ROOT	SHIFT
1	black	barrier in front of the person ROOT	LEFT(ATTR)
2		barrier in front of the person ROOT	SHIFT
3	barrier	in front of the person ROOT	SHIFT
4	barrier in	front of the person ROOT	LEFT(CONT)
5	barrier	front of the person ROOT	SHIFT
6	barrier front	of the person ROOT	LEFT(CONT)
7	barrier	of the person ROOT	SHIFT
8	barrier of	the person ROOT	SHIFT
9	barrier of the	person ROOT	REDUCE
10	barrier of	person ROOT	SHIFT
11	barrier of person	ROOT	RIGHT(OBJT)
12	barrier of	ROOT	RIGHT(SUBJ)
13	barrier	ROOT	LEFT(BEGN)
14		ROOT	

Figure 8.3. Intermediate actions taken by the trained dependency parser when parsing the sentence "black barrier in front of the person".

	Development set			Test set		
	R@5	**R@10**	**Med. rank**	**R@5**	**R@10**	**Med. rank**
[219]	33.82%	45.58%	6	34.96%	45.68%	**5**
Ours	**36.69%**	**49.41%**	**4**	**36.70%**	**49.37%**	**5**

Table 8.3. Image retrieval results. We follow the same experiment setup as [219], except using a different scoring function when ranking images. Our parser consistently outperforms the Stanford Scene Graph Parser across evaluation metrics.

8.5.3 Application in Image Retrieval

We test if the advantage of our parser can be propagated to computer vision tasks, such as image retrieval. We directly compare our parser with the Stanford Scene Graph Parser [219] on the development set and test set of the image retrieval dataset used in [219] (not Visual Genome).

For every region in an image, there is a human-annotated region description and region scene graph. The queries are the region descriptions. If the region graph corresponding to the query is a subgraph of the complete graph of another image, then that image is added to the ground truth set for this query. All these are strictly following [219]. However, since we did not obtain nor reproduce the CRF model used in [114] and [219], we used F-score similarity instead of the likelihood of the maximum a posteriori CRF solution when ranking the images based on the region descriptions. Therefore the numbers we report in Table 8.3 are not directly comparable with those reported in [219].

Our parser delivers better retrieval performance across all three evaluation metrics: recall@5, recall@10, and median rank. We also notice that the numbers in our retrieval setting are higher than those (even with oracle) in [219]'s retrieval setting. This strongly suggests that generating accurate scene graphs from images is a very promising research direction in image retrieval, and grounding parsed scene graphs to bounding box proposals without considering visual attributes/relationships [114] is suboptimal.

8.6 Conclusion

In this chapter, we offer a new perspective and solution to the task of parsing scene graphs from textual descriptions. We begin by moving the labels/types from the nodes to the edges and introducing the edge-centric view of scene graphs. We further show that the gap between edge-centric scene graphs and dependency parses can be filled with a careful redesign of label and action space. This motivates us to train a single, customized, end-to-end neural dependency parser for this task, as opposed to prior approaches that used generic dependency parsing followed by heuristics or simple classifier. We directly train our parser on a subset of Visual Genome [131], without transferring any knowledge from Penn Treebank [172] as previous works did. The quality of our trained parser is validated in terms of both SPICE similarity to the ground truth graphs and recall rate/median rank when performing image retrieval.

We hope this chapter can lead to more thoughts on the creative uses and extensions of existing NLP tools to tasks and datasets in other domains. In the future, we plan to tackle more computer vision tasks with this improved scene graph parsing technique in hand, such as image region grounding. We also plan to investigate parsing scene graph with cyclic structures, as well as whether/how the image information can help boost parsing quality.

Part III

Diagnosing 2D Image Recognition with 3D Objects

Chapter 9

Adversarial Attacks Beyond the Image Space

This chapter studies the robustness of 2D image recognition models against perturbations in 3D physical properties.

9.1 Introduction

Recent years have witnessed a rapid development in the area of deep learning, in which deep neural networks have been applied to a wide range of computer vision tasks, such as image classification [134][89], object detection [211], semantic segmentation [223][30], visual question answering [5][112], *etc*. Despite the great success of deep learning, there still lacks an effective method to understand the working mechanism of deep neural networks. An interesting effort is to generate so-called *adversarial perturbations*. They are visually imperceptible noise [78] which, after being added to an input image, changes the prediction results completely, sometimes ridiculously. These examples can be constructed in a wide range of vision problems, including image classification [189], object detection and semantic segmentation [267]. Researchers believed that the existence of adversaries implies unknown properties in the feature space [243].

Our work is motivated by the fact that conventional 2D adversaries were often generated by modifying each image pixel individually. We instead consider perturbations of the

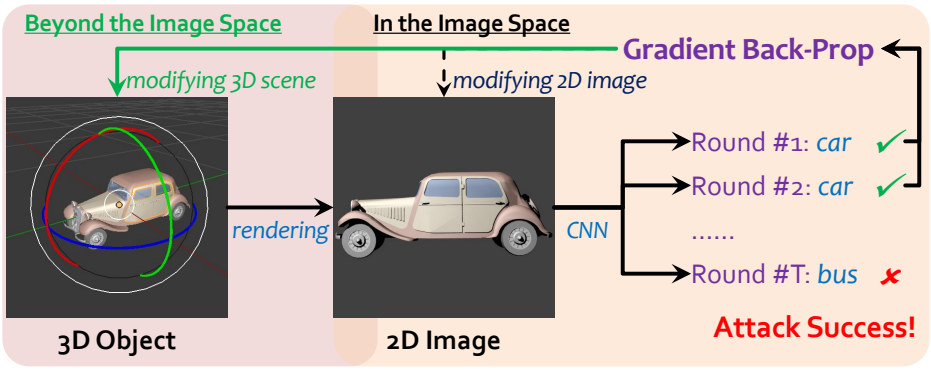


Figure 9.1. The vast majority of existing works on adversarial attacks focus on modifying pixel values in 2D images to cause wrong CNN predictions. In our work, we consider the more complete vision pipeline, where 2D images are in fact projections of the underlying 3D scene. This suggests that adversarial attacks can go *beyond* the image space, and directly change physically meaningful properties that define the 3D scene. We suspect that these adversarial examples are more physically plausible and thus pose more serious security concerns.

3D scene that are often non-local and correspond to physical properties of the object. We notice that previous work found adversarial examples "in the physical world" by taking photos on the printed perturbed images [137]. But our work is different and more essential, as we are attacking the intrinsic parameters that define the 3D scene/object, whereas [137] is still limited to attacking 2D image pixels. For this respect, we plug 3D rendering as a network module into the state-of-the-art neural networks for object classification and visual question answering. In this way, we build a mapping function from the *physical space* (a set of physical parameters, including surface normals, illumination and material), via the *image space* (a rendered 2D image), to the *output space* (the object class or the answer to a question). See Figure 9.1 which illustrates this framework.

The per-pixel image-space attack can be explained in terms of per-pixel changes of albedo, but it is highly unlikely that these individual perturbations happen to correspond to, e.g., a simple rotation of the object in 3D. Using our pipeline with rendering, we indeed found it almost impossible to approximate the 2D image adversaries using the 3D physically meaningful perturbations. At the same time, this suggests a natural mechanism

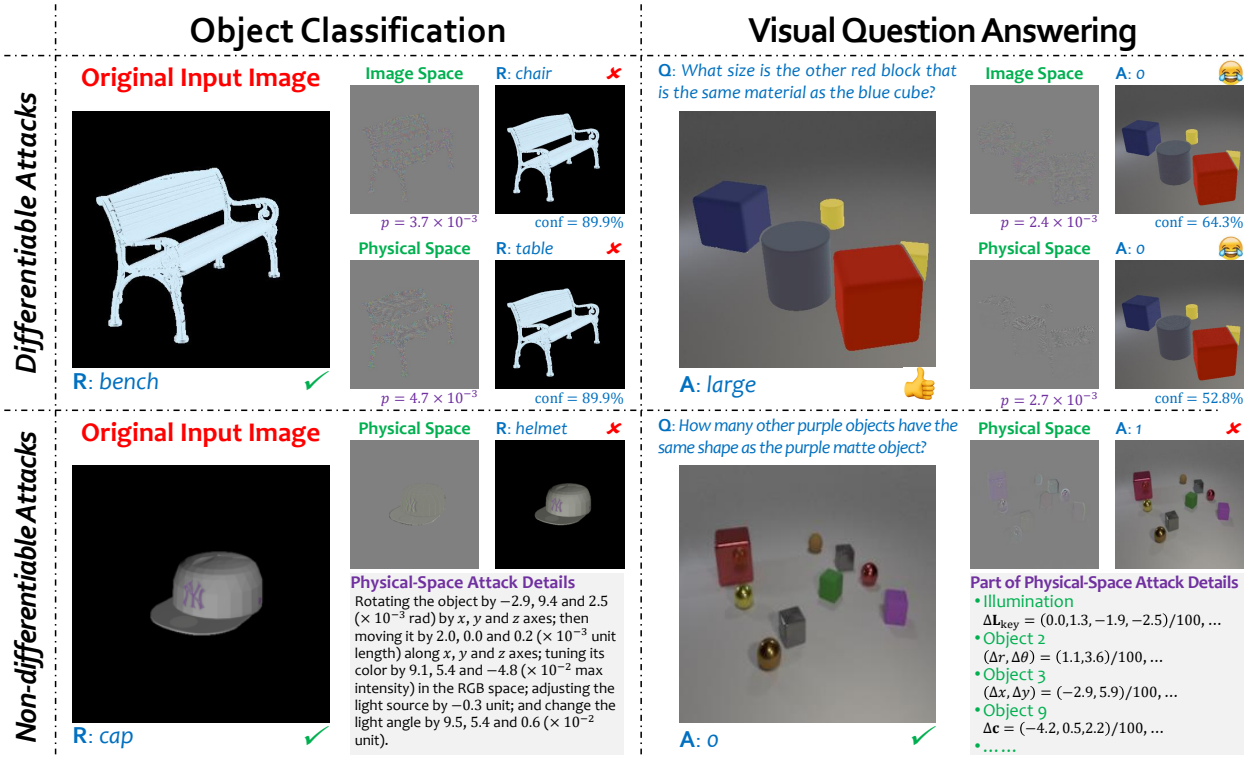


Figure 9.2. Adversarial examples for 3D object classification and visual question answering, under either a differentiable or a non-differentiable renderer. The top row shows that while it is of course possible to produce adversarial examples by attacking the image space, it is also possible to successfully attack on the physical space by changing factors such as surface normal, material, lighting condition (see Section 9.3.1). The bottom row demonstrates the same using a more realistic non-differentiable renderer, with descriptions of how to carry out the attack. p and conf are the perceptibility (see Section 9.3.2) and the confidence (post-softmax output) on the predicted class.

for defending adversaries – finding an approximate solution in the physical space and re-rendering will make most image-space adversaries fail. This analysis-by-synthesis process offers new direction in dealing with adversarial examples and occlusion cases.

This chapter mainly tries to answer the following question: **can neural networks still be fooled if we do not perturb 2D image pixels, but instead perturb 3D physical properties?** This is about directly generating perturbations in the physical space (*i.e.*, modifying basic physical parameters) that cause the neural network predictions to fail. Specifically, we compute the difference between the current output and the desired output, and use

gradient descent to update parameters in the physical space (*i.e.*, *beyond* the image space, which contains physical parameters such as surface normals and illumination conditions). This attack is implemented by either iterative Fast Gradient Sign Method (FGSM) [78] (for differentiable rendering) or the Zeroth-Order Optimization approach [33] (for non-differentiable rendering). We constrain the change in the image intensities to guarantee the perturbations to be visually imperceptible. Our major finding is that attacking the physical space is more difficult than attacking the image space. Although it is possible to find adversaries in this way (see Figure 9.2 for a few of these examples), the success rate is lower and the perceptibility of perturbations becomes much larger than required in the image space. This is expected, as the rendering process couples changes in pixel values, *i.e.*, modifying one physical parameter (*e.g.*, illumination) may cause many pixels to be changed at the same time.

9.2 Related Work

Deep learning is the state-of-the-art machine learning technique to learn visual representations from labeled data. Yet despite the success of deep learning, it remains challenging to explain what is learned by these complicated models. One of the most interesting evidence is *adversaries* [78]: small noise that is (i) imperceptible to humans, and (ii) able to cause deep neural networks make wrong predictions after being added to the input image. Early studies were mainly focused on image classification [189][184]. But soon, researchers were able to attack deep networks for detection and segmentation [267], and also visual question answering [273]. Efforts were also made in finding universal perturbations which can transfer across images [183], as well as adversarial examples in the physical world produced by taking photos on the printed perturbed images [137].

Attacking a known network (both network architecture and weights are given, *a.k.a*, a white box) started with setting a goal. There were generally two types of goals. The first

one (a non-targeted attack) aimed at reducing the probability of the true class [189], and the second one (a targeted attack) defined a specific class that the network should predict [160]. After that, the error between the current and the target predictions was computed, and gradients back-propagated to the image layer. This idea was developed into a set of algorithms, including the Steepest Gradient Descent Method (SGDM) [184] and the Fast Gradient Sign Method (FGSM) [78]. The difference lies in that SGDM computed accurate gradients, while FGSM merely kept the sign in every dimension. The iterative version of these two algorithms were also studied [137]. In comparison, attacking an unknown network (*a.k.a.*, a black box) is much more challenging [160], and an effective way is to sum up perturbations from a set of white-box attacks [267]. In opposite, there exist efforts in protecting deep networks from adversarial attacks [197][138][250]. People also designed algorithms to hack these defenders [23] as well as to detect whether adversarial attacks are present [177]. This competition has boosted both attackers and defenders to a higher level [6].

More recently, there is increasing interest in adversarial attacks other than modifying pixel values. [137] showed that the adversarial effect still exists if we print the digitally-perturbed 2D image on paper. [59][199] fooled vision systems by rotating the 2D image or changing its brightness. [62][7] created real-world 3D objects, either by 3D printing or applying stickers, that consistently cause perception failure. However, these adversaries have high perceptibility and must involve sophisticated change in object appearance. To find adversarial examples in 3D, we use a renderer, either differentiable or non-differentiable, to map a 3D scene to a 2D image and then to the output. In this way it is possible, though challenging, to generate interpretable and physically plausible adversarial perturbations in the 3D scene.

9.3 Approach

9.3.1 From Physical Parameters to Prediction

As the basis of this work, we extend deep neural networks to receive the physical parameters of a 3D scene, render them into a 2D image, and output prediction, *e.g.*, the class of an object, or the answer to a visual question. Note that our research involves 3D to 2D rendering as part of the pipeline, which stands out from previous work which either worked on rendered 2D images [241][113], or directly processed 3D data without rendering them into 2D images [205][221].

We denote the physical space, image space and output space by $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$, respectively. Given a 3D scene $\mathbf{X} \in \mathcal{X}$, the first step is to render it into a 2D image $\mathbf{Y} \in \mathcal{Y}$, and the second step is to predict the output of $\mathbf{Y}$, denoted by $\mathbf{Z} \in \mathcal{Z}$. The overall framework is denoted by $\mathbf{Z} = \mathbf{f}[\mathbf{r}(\mathbf{X}); \boldsymbol{\theta}]$, where $\mathbf{r}(\cdot)$ is the renderer, $\mathbf{f}[\cdot; \boldsymbol{\theta}]$ is the target deep network with $\boldsymbol{\theta}$ being parameters.

There are different models for the **3D rendering function** $\mathbf{r}(\cdot)$. One of them is *differentiable* [155], which considers three sets of physical parameters, *i.e.*, surface normals $\mathbf{N}$, illumination $\mathbf{L}$, and material $\mathbf{m}$[1]. By giving these parameters, we assume that the camera geometries, *e.g.*, position, rotation, field-of-view, *etc.*, are known beforehand and will remain unchanged in each case. The rendering module is denoted by $\mathbf{Y} = \mathbf{r}(\mathbf{N}, \mathbf{L}, \mathbf{m})$. In practice, the rendering process is implemented as a network layer, which is differentiable to input parameters $\mathbf{N}$, $\mathbf{L}$ and $\mathbf{m}$. Another option is to use a *non-differentiable* renderer which often provides much higher quality [16][175]. In practice we choose an open-source software named Blender [16]. Not assuming differentiability makes it possible to work on

[1] In this model, $\mathbf{N}$ is a 2-channel image of spatial size $W_\mathrm{N} \times H_\mathrm{N}$, where each pixel is encoded by the azimuth and polar angles of the normal vector at this position; $\mathbf{L}$ is defined by an HDR environment map of dimension $W_\mathrm{L} \times H_\mathrm{L}$, with each pixel storing the intensity of the light coming from this direction (a spherical coordinate system is used); and $\mathbf{m}$ impacts image rendering with a set of bidirectional reflectance distribution functions (BRDFs) which describe the point-wise light reflection for both diffuse and specular surfaces [190]. The material parameters used in this chapter come from the directional statistics BRDF model [191], which represents a BRDF as a combination of D_m distributions with P_m parameters in each. Mathematically, we have $\mathbf{N} \in \mathbb{R}^{W_\mathrm{N} \times H_\mathrm{N} \times 2}$, $\mathbf{L} \in \mathbb{R}^{W_\mathrm{L} \times H_\mathrm{L}}$ and $\mathbf{m} \in \mathbb{R}^{D_\mathrm{m} \times P_\mathrm{m}}$.

a wider range of parameters, such as color (**C**), translation (**T**), rotation (**R**) and lighting (**L**) considered in this work, in which translation and rotation cannot be implemented by a differentiable renderer[2].

We consider two popular **object understanding tasks**, namely, 3D object classification and 3D visual question answering, both of which are straightforward based on the rendered 2D images. Object classification is built upon standard deep networks, and visual question answering, when both the input image $\mathbf{Y}$ and question $\mathbf{q}$ are given, is also a variant of image classification (the goal is to choose the correct answer from a pre-defined set of choices).

In the adversary generation stage, given pre-trained networks, the goal is to attack a model $\mathbf{Z} = \mathbf{f}[\mathbf{r}(\mathbf{X}); \boldsymbol{\theta}] = \mathbf{f} \circ \mathbf{r}(\mathbf{X}; \boldsymbol{\theta})$. For object classification, $\boldsymbol{\theta}$ is fixed network weights, denoted by $\boldsymbol{\theta}^{\mathrm{C}}$. For visual question answering, it is weights from an assembled network determined by the question $\mathbf{q}$, denoted by $\boldsymbol{\theta}^{\mathrm{V}}(\mathbf{q})$. $\mathbf{Z} \in [0,1]^K$ is the output, with K being the number of object classes or choices.

9.3.2 Attacks Beyond the Image Space

Attacking the physical parameters starts with setting a *goal*, which is what we hope the network to predict. This is done by minimizing a loss function $\mathcal{L}(\mathbf{Z})$, which determines how far the current output is from the desired status. An adversarial attack may either be targeted or non-targeted, and in this work we focus on the non-targeted attack, which specifies a class c' (usually the original true class) as which the image should *not* be classified, and the goal is to minimize the c'-th dimension of the output $\mathbf{Z}$: $\mathcal{L}(\mathbf{Z}) \doteq \mathcal{L}(\mathbf{Z}; c') = Z_{c'}$.

[2] For 3D object classification, we follow [241] to configure the 3D scene. **L** is a 5-dimensional vector, where the first two dimensions indicate the magnitudes of the environment and point light sources, and the last three the position of the point light source. **C**, **T**, **R** are all 3-dimensional properties of the single object. For 3D visual question answering we follow [112]. **L** is a 12-dimensional vector that represents the energy and position of 3 point light sources. For every object in the scene, **C** is 3-dimensional, corresponding to RGB; **T** is 2-dimensional which is the object's 2D location on the plane; **R** is a scalar rotation angle.

An obvious way to attack the physical space works by expanding the loss function $\mathcal{L}(\mathbf{Z})$, *i.e.*, $\mathcal{L}(\mathbf{Z}) = \mathcal{L} \circ \mathbf{f} \circ \mathbf{r}(\mathbf{X}; \boldsymbol{\theta})$, and minimizing this function with respect to the physical parameters $\mathbf{X}$. The optimization starts with an initial (unperturbed) state $\mathbf{X}_0 \doteq \mathbf{X}$. A total of $T_{\max}$ iterations are performed. In the t-th round, we compute the gradient vectors with respect to $\mathbf{X}_{t-1}$, *i.e.*, $\Delta\mathbf{X}_t = \nabla_{\mathbf{X}_{t-1}} \mathcal{L} \circ \mathbf{f} \circ \mathbf{r}(\mathbf{X}_{t-1}, \boldsymbol{\theta})$, and update $\mathbf{X}_{t-1}$ along this direction: $\mathbf{X}_t = \mathbf{X}_{t-1} + \eta \cdot \Delta\mathbf{X}_{t-1}$, where η is the *learning rate*. This iterative process is terminated if the goal of attacking is achieved or the maximal number of iterations $T_{\max}$ is reached. The accumulated perturbation over all T iterations is denoted by $\Delta\mathbf{X} = \eta \cdot \sum_{t=1}^{T} \Delta\mathbf{X}_t$.

The way of computing gradients $\Delta\mathbf{X}_t$ depends on whether $\mathbf{r}(\cdot)$ is differentiable. If so, this can be simply back-propagate gradients from the output space to the physical space. We follow the Fast Gradient Sign Method (FGSM) [78] to only preserve the sign in each dimension of the gradient vector. Otherwise, we apply zeroth-order optimization. To attack the d-th dimension in $\mathbf{X}$, we set a small value δ and approximate the gradient of $\mathbf{Z}$ by $\frac{\partial \mathcal{L}(\mathbf{Z})}{\partial X_d} \approx \frac{\mathcal{L} \circ \mathbf{f} \circ \mathbf{r}(\mathbf{X} + \delta \cdot \mathbf{e}_d) - \mathcal{L} \circ \mathbf{f} \circ \mathbf{r}(\mathbf{X} - \delta \cdot \mathbf{e}_d)}{2 \times \delta}$, where $\mathbf{e}_d$ is a D-dimensional vector with the d-th dimension set to be 1 and all the others to be 0. In general, every step of such update may randomly select a subset of all D dimensions for efficiency considerations, so our optimization algorithm is a form of stochastic coordinate descent. This is reminiscent of [33], where each step updates the values of a random subset of pixel values. Also following [33], we use the Adam optimizer [125] instead of standard gradient descent for its faster convergence.

9.3.3 Perceptibility

The goal of an adversarial attack is to produce a visually imperceptible perturbation, so that the network makes incorrect predictions after it is added to the original image. Given a rendering model $\mathbf{Y} = \mathbf{r}(\mathbf{X})$ and an added perturbation $\Delta\mathbf{X}$, the perturbation added to the rendered image is: $\Delta\mathbf{Y} = \mathbf{r}(\mathbf{X} + \Delta\mathbf{X}) - \mathbf{r}(\mathbf{X})$.

There are in general two ways of computing perceptibility. One of them works directly

on the rendered image, which is similar to the definition in [243][184]: $p \doteq p(\Delta\mathbf{Y}) = \left(\frac{1}{W_{\mathrm{N}} \times H_{\mathrm{N}}} \sum_{w=1}^{W_{\mathrm{N}}} \sum_{h=1}^{H_{\mathrm{N}}} \|\Delta\mathbf{y}_{w,h}\|_2^2\right)^{1/2}$, where $\mathbf{y}_{w,h}$ is a 3-dimensional vector representing the RGB intensities (normalized in $[0, 1]$) of a pixel. Similarly, we can also define the perceptibility values for each set of physical parameters: $p(\Delta\mathbf{N}) = \left(\frac{1}{W_{\mathrm{N}} \times H_{\mathrm{N}}} \sum_{w=1}^{W_{\mathrm{N}}} \sum_{h=1}^{H_{\mathrm{N}}} \|\Delta\mathbf{n}_{w,h}\|_2^2\right)^{1/2}$.

We take $p(\Delta\mathbf{Y})$ as the major criterion of *visual* imperceptibility. Because of continuity, this can guarantee that all physical perturbations are sufficiently small as well. An advantage of placing the perceptibility constraint on pixels is that it allows a fair comparison of the attack success rates between image space attacks and physical space attacks. It also allows a direct comparison between attacks on different physical parameters. One potential disadvantage of placing the perceptibility constraint on physical parameters is that different physical parameters have different units and ranges. For example, the value range of RGB is $[0, 255]$, whereas that of spatial translation is $(-\infty, \infty)$. It is not directly obvious how to find a common threshold for different physical parameters.

When using the differentiable renderer, in order to guarantee imperceptibility, we constrain the RGB intensity changes on the image layer. In each iteration, after a new set of physical perturbations are generated, we check all pixels on the re-rendered image, and any perturbations exceeding a fixed threshold $U = 18$ from the original image is *truncated*. Truncations cause the inconsistency between the physical parameters and the rendered image and risk failures in attacking. To avoid frequent truncations, we set the learning rate η to be small, which consequently increases the number of iterations needed to attack the network.

When using the non-differentiable renderer, we pursue an alternative approach by adding another term $\|\Delta\mathbf{Y}\|_2^2$ into the loss function (weighted by λ) [23], [33], such that optimization can balance between attack success and perceptibility.

9.3.4 Interpreting Image Space Adversaries in Physical Space

We do a reality check to confirm that image-space adversaries are almost never consistent with the non-local physical perturbations according to our (admittedly imperfect) rendering model. They are, of course, consistent with per-pixel changes of albedo.

We first find a perturbation $\Delta\mathbf{Y}$ in the image space, and then compute a perturbation in the physical space, $\Delta\mathbf{X}$, that corresponds to $\Delta\mathbf{Y}$. This is to set the optimization goal in the image space instead of the output space, though the optimization process is barely changed. Note that we are indeed pursuing interpreting $\Delta\mathbf{Y}$ in the physical space. Not surprisingly, as we will show in experiments, the reconstruction loss $\|\mathbf{Y} + \Delta\mathbf{Y} - \mathbf{r}(\mathbf{X} + \Delta\mathbf{X})\|_1$ does not go down, suggesting that approximations of $\Delta\mathbf{Y}$ in the physical space either do not exist, or cannot be found by the currently available optimization methods such as FGSM.

9.4 Experiments

9.4.1 3D Object Classification

3D object recognition experiments are conducted on the ShapeNetCore-v2 dataset [24], which contains 55 rigid object categories, each with various 3D models. Two popular deep neural networks are used: an 8-layer AlexNet [134] and a 34-layer deep residual network [89]. Both networks are pre-trained on the ILSVRC2012 dataset [214], and fine-tuned in our training set for 40 epochs using batch size 256. The learning rate is 0.001 for AlexNet and 0.005 for ResNet-34.

We experiment with both a differentiable renderer [155] and a non-differentiable renderer [16], and as a result there are some small differences in the experimental setup, despite the shared settings described above.

For the **differentiable renderer**, we randomly sample 125 3D models from each class, and select 4 fixed viewpoints for each object, so that each category has 500 training

images. Similarly, another randomly chosen 50×4 images for each class are used for testing. AlexNet and ResNet-34 achieve 73.59% and 79.35% top-1 classification accuracies, respectively. These numbers are comparable to the single-view baseline accuracy reported in [241]. For each class, from the correctly classified testing samples, we choose 5 images with the highest classification probabilities on ResNet-34, and filter out 22 of them which are incorrectly classified by AlexNet, resulting in a target set of 233 images. The attack algorithm is the iterative version of FGSM [78]. We use the SGD optimizer with momentum 0.9 and weight decay 10^{-4}, and the maximal number of iterations is 120. Learning rate is 0.002 for attacking image space, 0.003 for attacking illumination and material, and 0.004 for attacking surface normal.

For the **non-differentiable renderer**, we render images with an azimuth angle uniformly sampled from $[0, \pi)$, a fixed elevation angle of $\pi/9$ and a fixed distance of 1.8. AlexNet gives a 65.89% top-1 testing set classification accuracy, and ResNet-34 achieves an even higher number of 68.88%. Among 55 classes, we find 51 with at least two images correctly classified. From each of them, we choose the two correct testing cases with the highest confidence score and thus compose a target set with 102 images. The attack algorithm is ZOO [33] with $\delta = 10^{-4}$, $\eta = 2 \times 10^{-3}$ and $\lambda = 0.1$. The maximal number of iterations is 500 for AlexNet and 200 for ResNet-34.

9.4.1.1 Differentiable Renderer Results

First, we demonstrate in Table 9.1 that adversaries widely exist in the image space – as researchers have explored before [243][184], it is easy to confuse the network with small perturbations. In our case, the success rate is at or close to 100% and the perceptibility does not exceed 10^{-2}.

The next study is to find the correspondence of these image-space perturbations in the physical space. We tried the combination of 3 learning rates ($10^{-3}, 10^{-4}, 10^{-5}$) and 2 optimizers (SGD, Adam). However, for AlexNet, the objective (ℓ_1-distance) remains

Attacking Perturbations	Image		Surface N.		Illumination		Material		Combined	
	Succ.	p	Succ.	p	Succ.	p	Succ.	p	Succ.	p
On AlexNet	100.00	5.7	89.27	10.8	29.61	25.8	18.88	25.8	94.42	18.1
On ResNet-34	99.57	5.1	88.41	9.3	14.16	29.3	3.43	55.2	94.85	16.4

Table 9.1. Effect of white-box adversarial attacks on ShapeNet object classification. By *combined*, we allow the three sets of physical parameters to be perturbed jointly. **Succ.** denotes the success rate of attacks (%, higher is better), and p is the perceptibility value (unit: 10^{-3}, lower is better). All p values are measured in the image space, *i.e.*, they are directly comparable.

mostly constant; the malicious label after image-space attack is kept in only 8 cases, and in the vast majority cases, the original true label of the object is recovered. Therefore, using the current optimization method and rendering model, it is very difficult to find physical parameters that are approximately rendered into these image-space adversaries. This is expected, as physical parameters often have a non-local effect on the image.

Finally we turn to directly generating adversaries in the physical space. As shown in Table 9.1, this is much more difficult than in the image space – the success rate becomes lower and large perceptibility values are often observed on the successful cases. Typical adversarial examples generated in the physical space are shown in Figure 9.3. Allowing all physical parameters to be jointly optimized (*i.e.*, the *combined* strategy) produces the highest success rate. Among the three sets of physical parameters, attacking surface normals is more effective than the other two. This is expected, as using local perturbations is often easier in attacking deep neural networks [78]. The surface normal matrix shares the same dimensionality with the image lattice, and changing an element in the matrix only has very local impact on the rendered image. In comparison, illumination and material are both global properties of the 3D scene or the object, so tuning each parameter will cause a number of pixels to be modified, hence less effective in adversarial attacks.

We also examined truncation during the attack. For ResNet-34, on average, only 6.3, 1.6, 0 pixels were ever truncated for normal, illumination, material throughout the 120 iterations of attack. This number of truncation is relatively small comparing to the size of

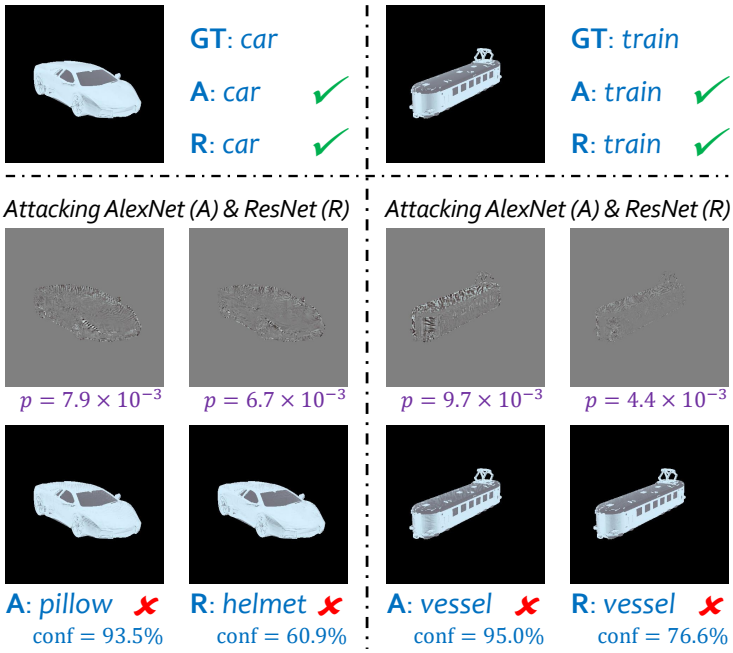


Figure 9.3. Examples of physical-space adversaries in 3D object classification on ShapeNet (using a differentiable renderer). In each example, the top row shows the original testing image, which is correctly classified by both AlexNet (A) and ResNet (R). The following two rows show the perturbations and the attacked image, respectively. All perturbations are magnified by a factor of **5** and shifted by **128**. p is the perceptibility value, and conf is the confidence (post-softmax output) of the prediction.

the rendered image (448×448). Therefore, the truncation is unlikely to contribute much to the attack.

9.4.1.2 Non-differentiable Renderer Results

We first report quantitative results with two settings, *i.e.*, attacking the image space and the physical space. Similarly, image-space adversaries are relatively easy to find. Among all 102 cases, 99 of them are successfully attacked within 500 steps on AlexNet, and all of them within 200 steps on ResNet-34. On the other hand, physical-space adversaries are much more difficult to construct. Using the same numbers of steps (500 on AlexNet and 200 on ResNet-34), the numbers of success attacks are merely 14 and 6 respectively.

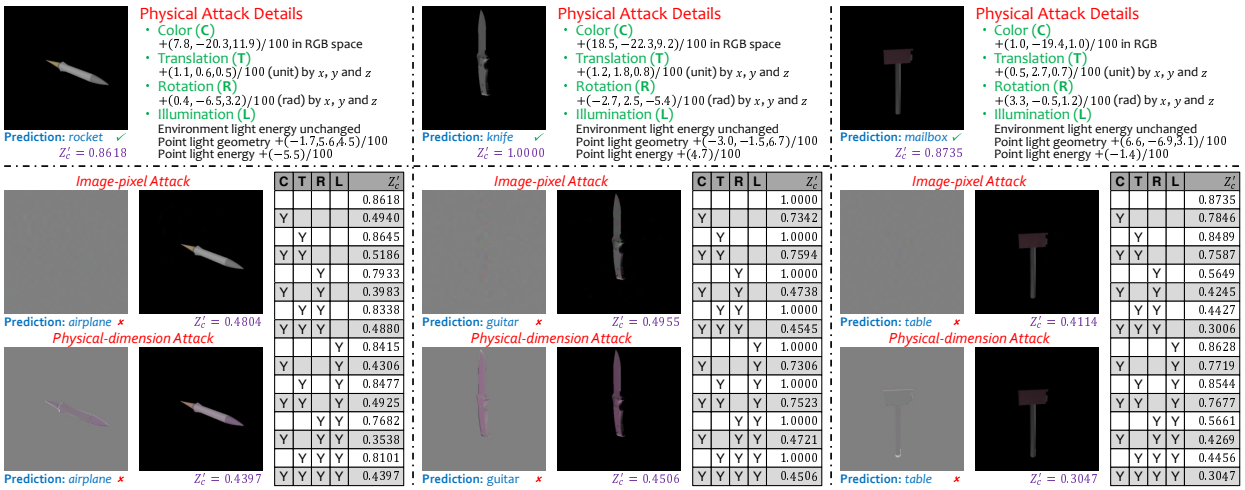


Figure 9.4. Examples of image-space and physical-space adversaries in 3D object classification on ShapeNet (using a non-differentiable renderer). In each example, the top row contains the original testing image and the detailed description of mid-level physical operations that can cause classification to fail. In the bottom row, we show the perturbations and attacked images in both attacks. Z'_c is the confidence (post-softmax output) of the true class. For each case, we also show results with different combinations of physical attacks in a table (a Y indicates the corresponding attack is on).

We show several successful cases of image-space and physical-space attacks in Figure 9.4. One can see quite different perturbation patterns from these two scenarios. An image-space perturbation is the sum of pixel-level differences, *e.g.*, even the intensities of two adjacent pixels can be modified individually, thus it is unclear if these images can really appear in the real world, nor can we diagnose the reason of failure. On the other hand, a physical-space perturbation is generated using a few mid-level operations such as slight rotation, translation and minor lighting changes. In theory, these adversaries can be instantiated in the physical world using a fine-level robotic controlling system.

Another benefit of generating physical-dimension adversaries lies in the ability of diagnosing vision algorithms. We use the cases shown in Figure 9.4 as examples. There are 14 changeable physical parameters, and we partition them into 4 groups, *i.e.*, the environment illumination (5 parameters), object rotation, position and color (3 parameters

each). We enumerate all 2^4 subsets of these parameters, and thus generate 2^4 perturbations by only applying the perturbations in the subsets. It is interesting to see that in the first case, the effects of different perturbations are almost additive, *e.g.*, the joint attack on color and rotation has roughly the same effect as the sum of individual attacks. However, this is not always guaranteed. In the second case, for example, we find that attacking rotation alone produces little effect, but adding it to color attack causes a dramatic accuracy drop of 26%. On the other hand, the second case is especially sensitive to color, and the third one to rotation, suggesting that different images are susceptible to attacks in different subspaces. It is the interpretability of the physical-dimension attacks that provides the possibility to diagnose these cases at a finer level.

9.4.2 Visual Question Answering

We extend our experiments to a more challenging vision task – visual question answering. Experiments are performed on the recently released CLEVR dataset [112]. This is an engine that can generate an arbitrary number of 3D scenes with meta-information (object configuration). Each scene is also equipped with multiple generated questions, *e.g.*, asking for the number of specified objects in the scene, or if the object has a specified property.

The baseline algorithm is named Inferring and Executing Programs (IEP) [113]. It applies an LSTM to parse each question into a tree-structure program, which is then converted into a neural module network [4] that queries the visual features. We use the released model without training it by ourselves. We randomly pick up 100 testing images, on which all associated questions are correctly answered, as the target images.

The settings for generating adversarial perturbations are the same as in the object classification experiments: when using the differentiable renderer, the iterative FGSM is used, and three sets of physical parameters are attacked either individually or jointly; when using the non-differentiable renderer, the ZOO algorithm [33] is used with $\delta = 10^{-3}$, $\eta = 10^{-2}$, $\lambda = 0.5$.

Attacking Perturbations	Image		Surface N.		Illumination		Material		Combined	
	Succ.	p	Succ.	p	Succ.	p	Succ.	p	Succ.	p
On IEP [113]	96.33	2.1	83.67	6.8	48.67	9.5	8.33	12.3	90.67	8.8

Table 9.2. Effect of white-box adversarial attacks on CLEVR visual question answering. By *combined*, we allow the three sets of physical parameters to be perturbed jointly. **Succ.** denotes the success rate of attacks (%, higher is better) of giving a correct answer, and p is the perceptibility value (unit: 10^{-3}, lower is better). All p values are measured in the image space, *i.e.*, they are directly comparable.

9.4.2.1 Differentiable Renderer Results

Results are shown in Table 9.2. We observe similar phenomena as in the classification experiments. This is expected, since after the question is parsed and a neural module network is generated, attacking either the image or the physical space is essentially equivalent to that in the classification task. Some typical examples are shown in Figure 9.5.

A side note comes from perturbing the material parameters. Although some visual questions are asking about the material (*e.g.*, *metal* or *rubber*) of an object, the success rate of this type of questions does not differ from that in attacking other questions significantly. This is because we are constraining perceptibility, which does not allow the material parameters to be modified by a large value.

A significant difference of visual question answering comes from the so called *language prior*. With a language parser, the network is able to clinch a small subset of answers without looking at the image, *e.g.*, when asked about the *color* of an object, it is very unlikely for the network to answer *yes* or *three*. Yet we find that sometimes the network can make such ridiculous errors. For instance, in the rightmost column of Figure 9.5, when asked about the *shape* of an object, the network answers *no* after a *non-targeted* attack.

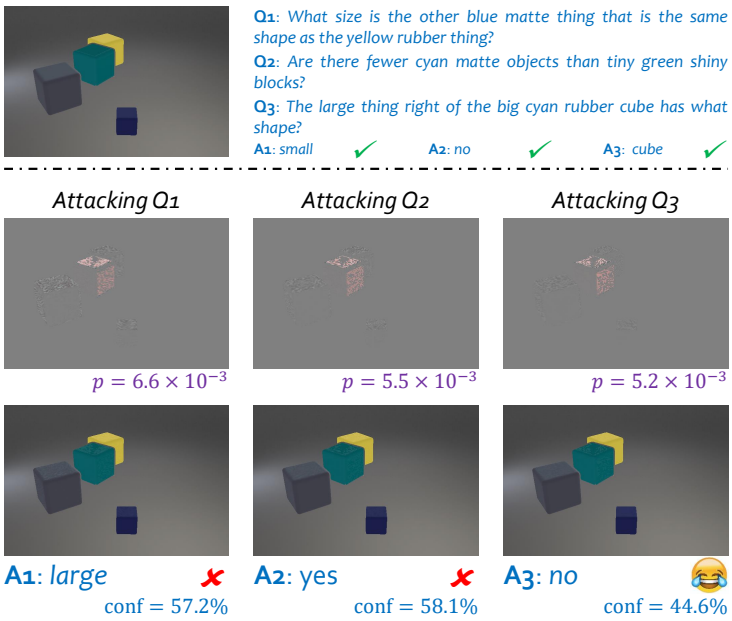


Figure 9.5. An example of physical-space adversaries in 3D visual question answering on CLEVR (using a differentiable renderer). In each example, the top row shows a testing image and three questions, all of which are correctly answered. The following two rows show the perturbations and the attacked image, respectively. All perturbations are magnified by a factor of **5** and shifted by **128**. p is the perceptibility value, and conf is the confidence (post-softmax output) of choosing this answer.

9.4.2.2 Non-differentiable Renderer Results

We observe quite similar results as in ShapeNet experiments. It is relatively easy to find image-space adversaries, as our baseline successfully attacks 66 out of 100 targets within 500 steps, and 93 within 1,200 steps. Due to computational considerations, we set 500 to be the maximal step in our attack experiment, but only find 22 physical-space adversaries. This is expected, since visual question answering becomes quite similar to classification after the question is fixed.

We show two successfully attacked examples in Figure 9.6. Unlike ShapeNet experiments, color plays an important role in CLEVR, as many questions are related to filtering/counting objects with specified colors. We find that in many cases, our algorithm

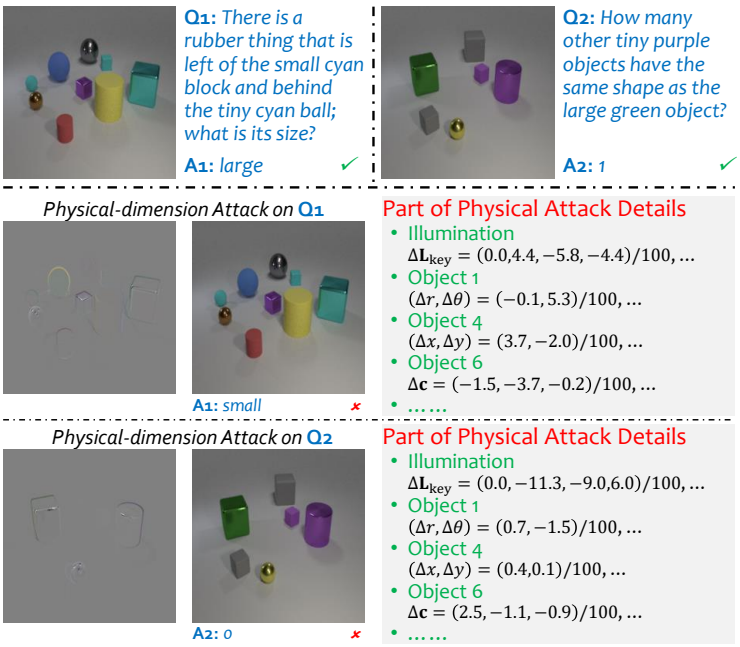


Figure 9.6. Examples of physical-space adversaries in 3D visual question answering on CLEVR (using a non-differentiable renderer). In each example, the top row contains a testing image and three questions. In the bottom row, we show the perturbations and attacked images. Detailed description of physical attacks on selective dimensions are also provided. All units of physical parameters follow the default setting in Blender.

achieves success by mainly attacking the color of the key object (*i.e.* that asked in the question). This could seem problematic, as generated adversaries may threaten the original correct answer. But according to our inspection, the relatively big λ we chose ensured otherwise. Nevertheless, this observation is interesting because our algorithm does not know the question (*i.e.*, IEP is a black-box) or the answer (*i.e.*, each answer is simply a class ID), but it automatically tries to attack the weakness (*e.g.*, color) of the vision system.

9.5 Conclusions

In this chapter, we generalize adversarial examples beyond the 2D image pixel intensities to 3D physical parameters. We are mainly interested to know: are neural networks vulnerable to perturbation on these intrinsic parameters that define a 3D scene, just like they are

vulnerable to artificial noise added to the image pixels?

To study this, we plug a rendering module in front of the state-of-the-art deep networks, in order to connect the underlying 3D scene with the perceived 2D image. We are then able to conduct gradient based attacks on this more complete vision pipeline. Extensive experiments in object classification and visual question answering show that directly constructing adversaries in the physical space is effective, but the success rate is lower than that in the image space, and much heavier perturbations are required for successful attacks. To the best of our knowledge, ours is the first work to study imperceptible adversarial examples in 3D, where each dimension of the adversarial perturbation has clear meaning in the physical world.

Going forward, we see three potential directions for further research. First, as a side benefit, our study may provide practical tools to diagnose vision algorithms, especially evaluating the robustness in some interpretable dimensions such as color, lighting and object movements. Second, in 3D vision scenarios, we show the promise to defend the deep neural networks against 2D adversaries by interpreting an image in the physical space, so that the adversarial effects are weakened or removed after re-rendering. Third, while our pipeline will continue to benefit from higher quality rendering, we also acknowledge the necessity to test out our findings in real-world scenarios.

Chapter 10

Identifying Model Weakness with Adversarial Examiner

This chapter describes an evaluation protocol that takes advantage of the better disentanglement in 3D than 2D to identify weaknesses of a trained model.

10.1 Introduction

The field of machine learning is advancing at unprecedented speed, and one evidence often cited is the reported performance on benchmark datasets. There are usually leaderboards for these public datasets with specific evaluation metrics, and the entry that achieves the highest number is regarded as the state-of-the-art. In several cases, it is claimed that the machine learning model has surpassed human-level performance using this criterion [88], [257].

However, the common belief is that humans are still superior, which suggests that the current testing strategy is overly optimistic and does not reflect the true progress in advancing machine learning. We think an important reason behind this mismatch is that: the current evaluation protocol uses *fixed* test data and measures the *average* case performance, which does not place enough emphasis on the *worst* case performance. To make this point concrete, imagine a model for autonomous driving. Suppose that in the testing data, there are 1 million images under normal conditions (in which case

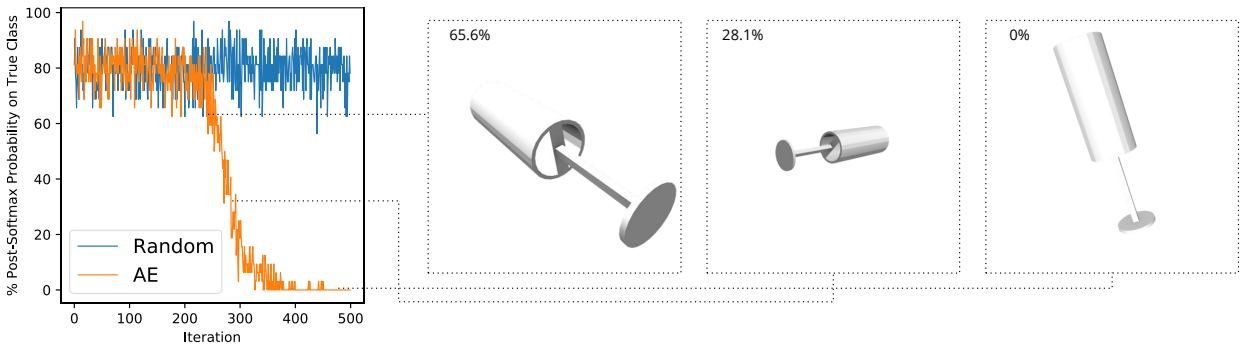


Figure 10.1. Evaluating a model's ability to recognize a *lamp* instance in ShapeNet. If we randomly sample data, the evaluation will converge to a high percentage (around 80%). However, using our Adversarial Examiner (AE), the test cases gradually concentrate on the weaknesses of the model, and the evaluation will not be overly optimistic. In this example, AE has found that the model appears to be vulnerable to increased lighting (within a reasonable range).

the car should keep driving), and 1 image with a baby in front (in which case the car should immediately stop). Under the current evaluation protocol, a policy to always keep driving and give up on the rare case will be highly regarded. However, from the humans' perspective, this strategy is clearly problematic and alarming. Therefore, at least in some sensitive domains, placing more emphasis on the worst cases to reveal model weaknesses may be more important than simply relying on the average case.

Revealing model weaknesses by densely sampling the input data space is clearly impractical. This is because any input data point is usually determined by a combination of factors, which makes the space exponentially large. For example, an image is jointly determined by the rotation angles of the objects, the positions of the objects, the material properties, the type of lighting, the energy of lighting, the distance to camera, etc. Larger test set certainly helps, but no matter how large it gets, it will never be able to densely cover this exponentially large space. Therefore our goal is to efficiently and systematically identify the model weaknesses within a reasonable number of queries.

Different models have different weaknesses, so it no longer makes sense to have a fixed test set, rather a dynamic one. In some way, this notion of fixed versus dynamic is very

similar to giving standardized tests versus conducting interviews. In standardized tests, the questions are usually designed to have general coverage, since all test takers answer the same questions. In interviews, the examiner may start with general topics, but also has the freedom to deep dive into specific ones to challenge the candidate. We also point out that this notion of dynamic examiner is closer in spirit to Turing test. If the human evaluator in Turing test always asks the same questions, then its role can be replaced by a machine. It is also conceivable that the worst answer during the exchange will contribute most to the human evaluator's final judgment.

Formalizing the ideas discussed above, we propose *adversarial examiner*, an alternative way of evaluating a model. Provided with a model and the space of input data, the adversarial examiner will dynamically select the next test data to hand out, based on the testing history so far. The name "adversarial" comes from the fact that its goal is to minimize the model's performance. Compared with the standard testing protocol, adversarial examiner obviously places more emphasis on the worst case scenarios. Because of its dependence on testing history, the sequence of test data naturally reveals the process of identifying model weakness. Figure 10.1 offers a demonstration.

Adversarial attack is also a way of showing the weakness of the model. The message of adversarial attack is that for a particular data point, there exists another data point within its local neighborhood, such that the model fails. However, both the original data point and the local neighborhood are microscopic properties. By contrast, our adversarial examiner starts with the entire input data space, and therefore has the potential of revealing the global picture. In addition, since the adversarial examiner concerns the global structure, the testing results are more informative and more easily perceived as "constructive feedback" that help improve the model in the next iteration.

We conduct experiments on ShapeNet object classification. The input data space is the Cartesian product of 12 physical parameters that determine the scene, including the rotation and position of object, the energy and angle of lighting, etc. Through both quanti-

tative evaluation and qualitative visualization, we show that our adversarial examiner can successfully identify the weakness of the model, preventing performance estimates from being overly optimistic.

10.2 Method

10.2.1 Adversarial Examiner

We begin by revisiting the evaluation protocol employed in a standard classification task. The examples $x \in \mathbb{R}^p$ follow an underlying distribution $\mathcal{P}$. The ground truth label of x is $y(x) \in \{1, 2, \ldots, k\}$. The label predicted by the model f is $f(x) \in \mathbb{R}^k$. Given a loss function $L(\cdot, \cdot)$ that measures the compatibility between $f(x)$ and $y(x)$, the evaluation metric is:

$$E = \mathbb{E}_{x \sim \mathcal{P}}[L(f(x), y(x))] \approx \frac{1}{N} \sum_{i=1}^{N} L(f(x_i), y(x_i)) \tag{10.1}$$

where $x_i \sim \mathcal{P}$.

In applications such as computer vision and natural language processing, the examples x are usually the surface representation, e.g. 2D images in vision and words in language. However, there is typically an underlying representation z that generates/determines this surface representation x. In vision, z would be the 3D object instance. In language, z would be the semantics/pragmatics that the speaker wishes to communicate. In the following, we consider evaluation on the underlying representation z, which is more intrinsic.

The examples $z \in \mathbb{R}^q$ follow an underlying distribution $\mathcal{Q}$, whose support is $\mathcal{D}$. If we can obtain the ground truth label using the surface form x, then we can certainly obtain it using the underlying form z. We slightly abuse annotation and use $y(z) \in \{1, 2, \ldots, k\}$ to represent the ground truth label for instance z. Let g be the function that transforms the underlying form into surface form $x = g(z, s)$, where $s \in \mathcal{S}$ represents the remaining information needed to complete this transformation. For example, in vision, s could include camera information, lighting condition, etc. In language, s may be a tie-breaker

between synonyms or sentence templates that express the same meaning.

We use the following evaluation metric in our *adversarial examiner*:

$$\begin{aligned} E_{\text{examiner}} &= \mathbb{E}_{z\sim\mathcal{Q}}[\max_{s\in\mathcal{S}} L(f(g(z,s)),y(z))] \\ &\approx \frac{1}{N}\sum_{i=1}^{N}\max_{s_i\in\mathcal{S}} L(f(g(z_i,s_i)),y(z_i)) \end{aligned} \tag{10.2}$$

where $z_i \sim \mathcal{Q}$. We choose $\mathcal{S}$ such that for any instance $z \in \mathcal{D}$ and any $s \in \mathcal{S}$, the surface form $g(z,s)$ still preserves the label $y(z)$ as judged by humans. The max in (10.2) means the examiner will consider the worst case within the space $\mathcal{S}$, which differs from (10.1).

It is easy to see that

$$E_{\text{examiner}} \leq \max_{z\in\mathcal{D},s\in\mathcal{S}} L(f(g(z,s)),y(z)) = L(f(g(\hat{z},\hat{s})),y(\hat{z})) \tag{10.3}$$

where $(\hat{z},\hat{s})$ is the worst case scenario. We now discuss a special case, where the space $\mathcal{S}$ is so rich, that $\forall z_i, z_j \in \mathcal{D}$, $\exists s_i, s_j \in \mathcal{S}$, such that $g(z_i,s_i) = g(z_j,s_j)$. In this case, the worst combination $(\hat{z},\hat{s})$ can always be achieved from any sample z_i. Therefore, E_{examiner} will return *exactly* the worst case performance under model f, which is what we expect and desire.

Implementing the adversarial examiner according to (10.2) requires solving an optimization problem for every instance z_i. Since the landscape of $f(g(z_i,\cdot))$ on $\mathcal{S}$ can be arbitrary, we address the general case by assuming no closed-form solution and no differentiability. We consider a sequential strategy, where the adversarial examiner will hand out new candidates of s_i^t based on the testing history so far $s_i^1, s_i^2, \ldots, s_i^{t-1}$. See Algorithm 10.1 for the adversarial examiner procedure.

Relation to Adversarial Attacks We named our examiner "adversarial" to acknowledge its connection to adversarial attacks. If we conduct adversarial attacks on the samples x_i

Algorithm 10.1: Adversarial Examiner Procedure

Input: N samples $z_i \sim \mathcal{Q}$ and their true labels $y(z_i)$; Maximum number of examination steps T; Loss function L; Model f; Function g; Space $\mathcal{S}$.

for $i = 1$ **to** N **do**
  Initialize `examiner` with $\mathcal{S}$
  for $t = 1$ **to** T **do**
    $s_i^t =$ `examiner.generate()`
    $l_i^t = L(f(g(z_i, s_i^t)), y(z_i))$
    `examiner.update`(s_i^t, l_i^t)
return $E_{\text{examiner}} = \frac{1}{N}\sum_{i=1}^{N} l_i^T$

before evaluating using (10.1), the result is:

$$E_{\text{attack}} \approx \frac{1}{N}\sum_{i=1}^{N} \max_{\delta_i \in \Delta} L(f(x_i + \delta_i), y(x_i)) \tag{10.4}$$

There are at least two key differences between adversarial attack (AA) and adversarial examination (AE). First, AE deals with the underlying form whereas AA deals with the surface form. As a result, the variations considered by AE on the surface form is much larger than Δ. Second and more importantly, in AA there is a "canonical" starting point $\delta_i = 0$, so essentially all attack algorithms perform gradient descent starting from $\delta_i = 0$. As a result, the sequence $\delta_i^1, \delta_i^2, \ldots$ is usually very local. In AE, there is typically no "canonical" choice for s_i, so AE is forced to start with the entire space $\mathcal{S}$ instead of a specific point. As a result, the sequence $s_i^1, s_i^2, \ldots$ exhibits large variations. This global picture reveals different properties of f than AA, and is potentially more useful in representing the weakness (and/or strength) of the model.

Availability of Underlying Representation While data in the surface form are straightforward to collect, there may be concerns whether it is always easy to obtain the underlying representation z considered by the adversarial examiner. We argue that this problem has been greatly alleviated in recent years, with either rich annotation on real data [131], [265], or increasing use of synthetic data [24], [112], or a hybrid between real and synthetic [70],

[107]. In addition, in related topics such as image caption evaluation, the advantage of evaluating the underlying representation (e.g. SPICE [3]) over the surface representation (e.g. BLEU [198]) has been demonstrated.

10.2.2 Model Choices for Examiner

We describe two classes of models as the `examiner` in Algorithm 10.1. One is based on Reinforcement Learning (Section 10.2.2.1), and the other is based on Bayesian Optimization (Section 10.2.2.2). We conclude by discussing how these two classes of models are complementary (Section 10.2.2.3), which shows the general applicability of our adversarial examiner framework.

10.2.2.1 Reinforcement Learning as Examiner

Let space $\mathcal{S}$ be the Cartesian product of C factors $\mathcal{S} = \Psi^1 \times \Psi^2 \times \cdots \times \Psi^C$, where Ψ^i represents the parameter range of the i-th factor. Therefore, the candidate s_i^t is composed of $\psi_{(i,t)}^1, \psi_{(i,t)}^2, \ldots, \psi_{(i,t)}^C$, where $\psi_{(i,t)}^c \in \Psi^c$. The probability of generating s_i^t is $P(s_i^t) = \prod_{c=1}^C P(\psi_{(i,t)}^c | \psi_{(i,t)}^{c-1:1})$. We use a LSTM [92] to parameterize these conditional probabilities. Concretely, in the first LSTM step, the initial hidden state h^1 is followed by a fully connected layer and softmax to represent $P(\psi_{(i,t)}^1)$. We then draw a sample according to this distribution, which is fed into the second LSTM step. The updated hidden state h^2 is followed by a fully connected layer and softmax to represent $P(\psi_{(i,t)}^2 | \psi_{(i,t)}^1)$. This process is repeated until we have drawn a complete sample of C steps.

To train this LSTM using reinforcement learning, we define the reward signal R to be $L(f(g(z_i, s_i^t)), y(z_i))$, and optimize the weights θ using policy gradient [260]:

$$\nabla_\theta \mathbb{E}_{P(s_i^t;\theta)}[R] \approx \frac{1}{B} \sum_{b=1}^{B} \sum_{c=1}^{C} \nabla_\theta \log P(\psi_{(i,t)}^c | \psi_{(i,t)}^{c-1:1}) R_b \tag{10.5}$$

where B is the batch size to reduce high variance.

10.2.2.2 Bayesian Optimization as Examiner

We use Gaussian Process (GP) as the model behind Bayesian optimization. The value that the GP aims to maximize is $L(f(g(z_i, s_i^t)), y(z_i))$ from Algorithm 10.1. The example generated by our examiner is equivalent to the point proposed by the acquisition function $a : \mathcal{S} \rightarrow \mathbb{R}^+$. We use Gaussian Process upper confidence bound (UCB) [237] to construct our acquisition function.

Let W be the set of points that induce the posterior multivariate Gaussian distribution, which is initially empty. For each iteration $t = 1, 2, \ldots, T$, we select the next candidate by:

$$s_i^t = \underset{s \in \mathcal{S}}{\operatorname{argmax}}\, a(s) \tag{10.6}$$

where a is the acquisition function induced by the current W. We then construct its surface form and test on the model. This newly observed point $(s_i^t, L(f(g(z_i, s_i^t)), y(z_i)))$ is then added to set W. In the next iteration, the examiner will induce a new posterior multivariate Gaussian distribution based on the updated set W. By the end of adversarial examination, the candidates $\{s_i^t \in \mathcal{S}\}_{t=1}^T$ are the points that induce the most up-to-date posterior multivariate Gaussian distribution on $\mathcal{S}$.

10.2.2.3 How the Two Examiners Are Complementary

Discrete vs. Continuous In RL, the choices of parameters in Ψ are discrete as we use soft-max to select amongst them. As for BO, the ranges of parameters are typically continuous as the underlying assumption is a multivariate Gaussian distribution.

Maintaining Sampling Distribution on $\mathcal{S}$ vs. Maintaining Function Value on $\mathcal{S}$ The LSTM model within RL captures the sampling probability P on $\mathcal{S}$, without explicitly recording the function value L. On the other hand, the GP model within BO relies on the function values (set W) to fit the acquisition function.

Longer Iteration Regime vs. Shorter Iteration Regime BO can be quite sample efficient, and is potentially able to successfully attack the model within a limited number of steps. However, it becomes costly when the iteration gets long. On the other hand, policy-based RL can be inefficient, but the computation cost does not go up with more iterations.

10.3 Related Work

We argue that the average case performance does not always reflect human's level of trust in a model. The primary design philosophy behind adversarial examiner is to place more emphasis on the worst case performance, which especially matters in sensitive domains. Indeed, medical imaging papers typically report the worst performance on the test set in addition to the average performance. The idea of paying more attention to the worst cases was recently discussed in [285].

As suggested by its name, adversarial examiner is related to finding adversarial examples [243]. They were originally found by doing backpropagation and gradient ascent on the input image. Later researchers started to consider the more challenging but more general scenario, where the neural network weights are not known to the attacker. This is called black box adversarial attack, where existing methods [33], [110], [186] still rely on estimating the gradient. Essentially, black box adversarial attack belongs to derivative-free optimization, which includes many other families of methods than gradient approximation. To the best of our knowledge, ours is the first work that brings Reinforcement Learning (RL) and Bayesian Optimization (BO) to black box adversarial attack. Our RL setup is inspired by work on Neural Architecture Search [297], whereas our BO setup is inspired by work on hyperparameter selection [233]. Again, both problems are typical derivative-free optimization.

Recently, researchers have been rethinking the concept of adversarial examples [76] and generalizing them beyond modifying pixel values. [8], [59], [199] showed that deep

networks can be attacked simply by 2D rotation, translation, or brightness change. [7], [62], [137] demonstrated the possibility of generating physical adversarial examples in the real world. [84] used the minute transformations across video frames to study adversarial robustness. [18] introduced the concept of unrestricted adversarial examples, and [235] described a method to generate unrestricted adversarial examples using a differentiable neural network. More related to our work, [2], [118], [275], [287] incorporated rendering into the visual recognition pipeline to study how 3D physical parameters may affect model prediction. However, they were all concerned with creating adversarial examples starting from a specific configuration, instead of considering the global space.

Finally, there are a couple of works that used synthetic data to iteratively improve the model, whether in the training stage or the inference stage. [34] proposed to increase the sampling probability in the underperforming region. [181] used an agent to learn to select questions in synthetic visual question answering. [276] learned to navigate in a synthetic scene and select better views for visual recognition. [119] learned the policy to generate synthetic scenes, whose goal is to best facilitate downstream tasks. Our work is different, in that we focus on a new paradigm of model evaluation, which systematically explores the input data space to identify model weaknesses.

10.4 Experiments

10.4.1 Implementation Details

We conduct experiments on visual recognition of objects in the ShapeNet dataset [24], which contains 55 classes and 51190 instances. Here, the underlying representation z would be the 3D object, s would be parameters required in rendering (we consider a total of $C = 12$ factors, listed in Table 10.1), and surface form x would be the 2D image after rendering. We use the Blender software for rendering.

Our target models are ResNet34 [89] and AlexNet [134] trained on the 2D image surface

	α_o	β_o	ζ_o	Γ_o	Γ_l	r_l	A_l	U_l	r_v	A_v	U_v	θ_v
UB	2π	2π	2π	5	1	20	360	90	5	180	90	360
LB	0	0	0	0	0.3	8	0	-90	1	0	-90	0

Table 10.1. Upper bound (UB) and lower bound (LB) of rendering factors in s: sun rotation angles $(\alpha_o, \beta_o, \zeta_o)$, sun energy (Γ_o), point light energy (Γ_l), point light distance (r_l), point light location (A_l, U_l), viewpoint distance (r_v), viewpoint location (A_v, U_v), viewpoint angle (θ_v).

form x. The ResNet34 model is trained with learning rate of 0.005, and AlexNet model with 0.001, both with Adam optimizer [125] for 40 epochs.

During training, we randomly select a value for each of the 12 factors, and render $m = 10$ images per 3D object. During adversarial examination, we assume that the examiners have no control over the location of the sun, so we randomly choose the rotation angles for the sun and fix them. The remaining 9 factors will be available to and explored by the examiners. For each class, we choose one 3D object in the validation set that has the highest post-softmax probability on the true class.

In RL, the 9 continuous factors will be discretized to 100 choices evenly distributed on their respective ranges. When sampling the factors, the choice made will be mapped to its corresponding embedding space (embedding size 30), then fed into LSTM (hidden state size 30). We set the learning rate to 0.001 and batch size to 32, and use Adam optimizer [125] to update model parameters. In BO, our implementation is based on the `Bayesian Optimization` package[1] and we allow 2 random examples at the very beginning. Both examiners operate for $T = 500$ iterations, and L is the negative post-softmax probability on the true class.

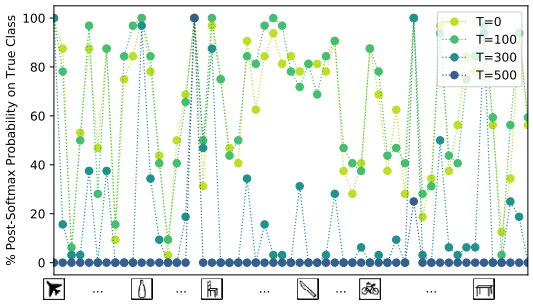
10.4.2 Evaluating Model Performance with Adversarial Examiners

The evaluation results by our adversarial examiners are shown in Table 10.2. For both AlexNet and ResNet34, we tried both RL and BO examiners under different number of

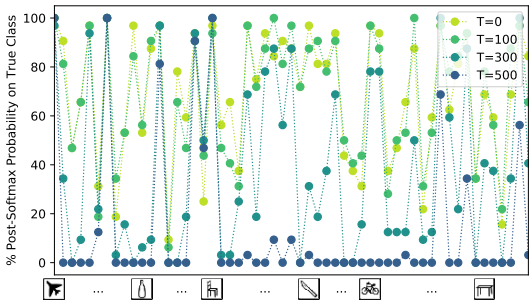
[1] `https://github.com/fmfn/BayesianOptimization`

Model	Examiner	$T = 0$	$T = 100$	$T = 300$	$T = 500$
AlexNet	RL	63.98%	65.91%	18.92%	2.27%
	BO	60.05%	43.58%	29.98%	25.43%
ResNet34	RL	69.03%	68.58%	38.86%	13.13%
	BO	64.19%	54.89%	48.07%	45.55%

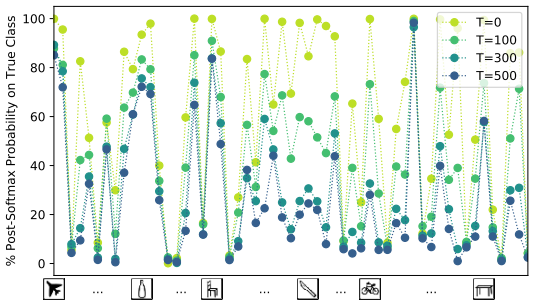
Table 10.2. Average model performance under adversarial examination. We report $2 \times 2 \times 4 = 16$ settings under various model, examiner, and number of iterations combinations. The values are average post-softmax probability on the true class.



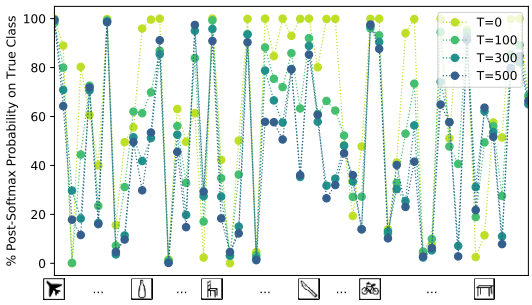
(a) RL Examiner on AlexNet



(b) RL Examiner on ResNet34



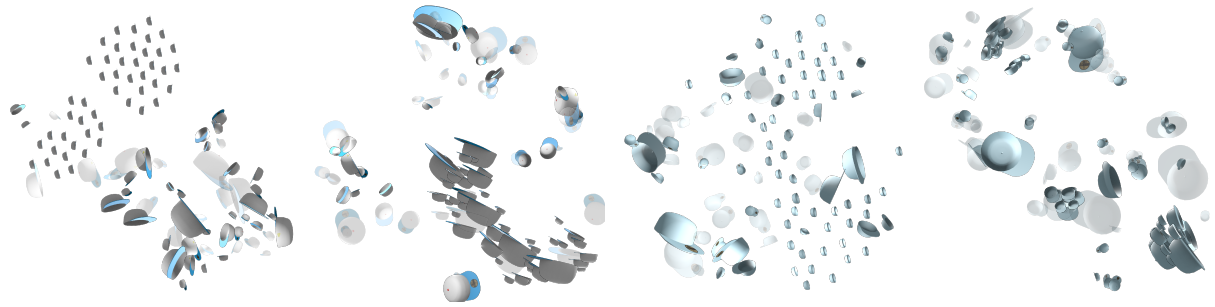
(c) BO Examiner on AlexNet



(d) BO Examiner on ResNet34

Figure 10.2. Per-class model performance under adversarial examination. The four plots correspond to AlexNet (left) vs. ResNet34 (right), RL (upper) vs. BO (lower). Horizontal axis is object category. AlexNet is more vulnerable than ResNet34, and the RL examiner seems more strict than BO examiner under the same T.

iterations T. The $T = 0$ column corresponds to the standard evaluation protocol, which on average records more than 60% probability on the true class. After a few hundred iterations,



(a) RL on AlexNet **(b)** BO on AlexNet **(c)** RL on ResNet34 **(d)** BO on ResNet34

Figure 10.3. t-SNE visualization of *cap* examples under different examiners and target models. There are 100 examples in each subfigure: 50 randomly sampled ones, and 50 from the very end of adversarial examination. Transparent cap means correctly classified. Otherwise, incorrectly classified.

both examiners can find the weakness of the model successfully. In addition, longer examination results in lower performance, which is expected. But their characteristics are not entirely the same. BO can find weaknesses relatively faster (larger gap from $T = 0$ to $T = 100$), but eventually ($T = 300$ or $T = 500$) RL becomes more committed to the weaknesses and gives a harsher evaluation.

Figure 10.2 shows the per-class post-softmax probability on the true class. We found that the weaknesses of some classes are easier to find than others. For instance, *barrel* and *bowl* are relatively vulnerable since they look very similar (generic round shape) when viewing from bottom-up. Classes such as *airplane*, which contain discriminative visual cues at all angles, are more robust under adversarial examination.

Figure 10.3 uses t-SNE [167] to visualize 50 randomly sampled examples and 50 examples selected at the end of adversarial examination of the *cap* class. Correctly classified examples are transparent, and incorrectly classified ones are opaque. This visualization serves to show the global distribution of model weakness on the space $\mathcal{S}$. We can observe clusters among the incorrectly classified examples, e.g. due to viewpoint or lighting.

	$m = 10$	$m = 5$	$m = 2$	$m = 1$
RL	63.81%	57.43%	35.05%	18.92%
BO	49.79%	43.06%	22.19%	10.92%

Table 10.3. Average model performance under adversarial examination for varying m: number of training images per instance. We report $m = 1, 2, 5, 10$ with iterations $T = 200$.

10.4.3 Examining Models Trained with Less Data

Intuitively, neural networks trained with less data should be less robust against adversarial examiners. In this experiment, we put this hypothesis to test. Specifically, we vary m, the number of images rendered for each 3D object in the training set. We train ResNet34 models with $m = 1, 2, 5, 10$ (recall that $m = 10$ is the default) using the same number of epochs.

Table 10.3 summarizes the results. Under the same iterations T, both RL and BO examiners report decreased performance as m decreases. In addition to validating the adversarial examiner framework, this also confirms the importance of the amount and diversity of training data.

10.4.4 Evaluating Model with Artificial Weaknesses

To further study the properties of adversarial examiner, we consider an artificial scenario, where we deliberately put in obvious weaknesses. We are interested to see to what extent can the adversarial examiner recover these obvious weaknesses. Specifically, during training, we limit the viewpoint elevation U_v to $[-30^\circ, +30^\circ]$ instead of $[-90^\circ, +90^\circ]$. This will create two obvious weaknesses: viewing objects from the top and from the bottom. Note that the examiner is still allowed to change all 9 factors.

Figure 10.4 shows the t-SNE visualization of 200 *can* examples. 100 are randomly sampled, and 100 are from $T = 250$ during adversarial examination. Among the incorrectly classified examples (visualized as opaque), there are two large regions, with the bottom

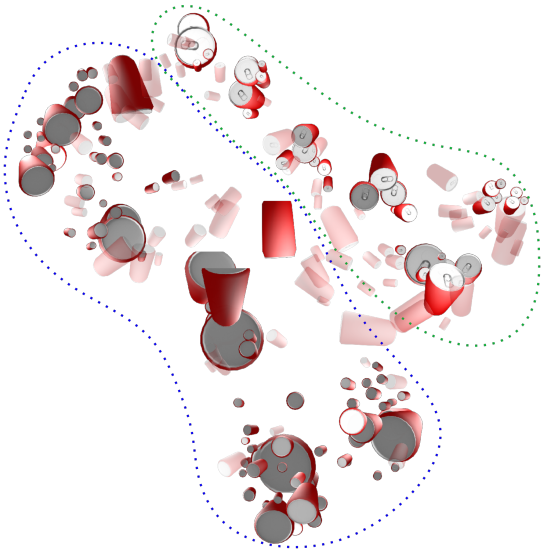


Figure 10.4. t-SNE visualization of *can* examples. 100 are randomly sampled, and 100 are from RL examiner $T = 250$. Transparent can means correctly classified. Otherwise, incorrectly classified. In this experiment, training data for ResNet34 did not have top and bottom viewpoints. Adversarial examiner is able to identify these two artificial weaknesses (circled in green and blue).

left representing viewing from the bottom, and the top right representing viewing from the top. This observation further confirms the capability of adversarial examiners.

10.4.5 Changing the Order of Factors

As described in Section 10.2.2.1, the RL examiner samples parameters in a sequential manner. In this experiment, we investigate whether changing the sampling order of factors will make a difference in adversarial examination.

The default RL examiner samples in the order $\Gamma_l, \Gamma_o, r_l, A_l, U_l, r_v, A_v, U_v, \theta_v$. We prepare a second RL examiner with exactly the same setup, but instead samples in the order $r_v, A_v, U_v, \theta_v, \Gamma_l, \Gamma_o, r_l, A_l, U_l$. Figure 10.5 shows the t-SNE visualization of the last 50 examples given by the two RL examiners ($T = 500$). Obviously they have converged to the same weaknesses, which is evidence that the RL examiner is not sensitive to the order.

(a) $\Gamma_l, \Gamma_o, r_l, A_l, U_l, r_v, A_v, U_v, \theta_v$ **(b)** $r_v, A_v, U_v, \theta_v, \Gamma_l, \Gamma_o, r_l, A_l, U_l$

Figure 10.5. The last 50 *lamp* examples given by two RL examiners with different order of factors.

10.4.6 Identifying Model Strength

Finally, we show that instead of identifying model weakness, the same setup can also be used to identify model strength, simply by flipping the max in (10.2) to min. The examiner, instead of trying to find the worst viewing condition of an object, now tries to find the best viewing condition.

We show such a trial in Figure 10.6. We indeed observe a curve where the post-

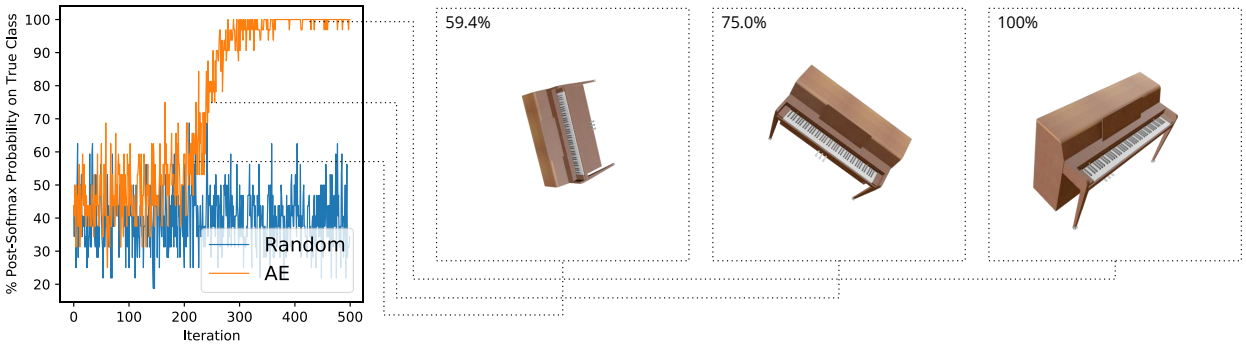


Figure 10.6. Identifying model strength by negating L. For this *piano* instance, the examiner eventually finds a condition under which the model can correctly classify with 100% confidence.

softmax probability on the true class goes up throughout evaluation, which is symmetric to Figure 10.1. The final image reveals what the model considers to be the easiest *piano* to recognize.

10.5 Conclusion

In this chapter, we advocate for a new testing paradigm for machine learning models, where more emphasis is placed on the worst case instead of reporting the average case performance. To make this idea concrete, we propose *adversarial examiner*, whose goal is to undermine the model's performance by discovering and concentrating on its weaknesses. The adversarial examiner will dynamically hand out the next testing data, based on the testing history so far. As a result, different models will now be evaluated using potentially different samples from the input data space. We argue that this notion can more effectively prevent hype, is closer in spirit to human interview/Turing test, and can be especially important in sensitive domains such as autonomous driving and medical applications.

We conduct experiments on visual recognition of ShapeNet objects. Examiners based on Reinforcement Learning and Bayesian Optimization both demonstrated effectiveness in identifying the weaknesses of the target model. However, our adversarial examination framework is much more general than classification of rendered 3D objects. In the future, we hope to extend to other domains, including image generation, language generation, and pose estimation in robotics. Using the examples generated during adversarial examination to improve training is another potentially fruitful venue.

Chapter 11

Conclusion

11.1 Summary

In this dissertation, I present a total of nine research projects. Though they are concerned with various components of the visual intelligence pipeline, they share the common goal of either improving visual perception, or making this process more interpretable, or both.

Part I aims at designing better models for image recognition, which is a core step that converts signals to symbols and often benefits a wide range of downstream tasks. The route I am taking is to let machines do the heavy lifting in this designing process (typically named AutoML or NAS), as opposed to relying solely on humans. Although this usually means longer time and more computation, the upside is potentially deeper understanding and higher ceiling, much like the difference between AlphaGo Zero [231] and AlphaGo [230]. Each chapter creates a new dimension on top of the previous one. Chapter 2 presents an effective method to "factorize" the search space and significantly speed up the designing process. Chapter 3 describes a minimal and elegant way to extend NAS from image classification to dense image prediction problems. Chapter 4 generalized NAS from the supervised setting to the unsupervised setting, highlighting the importance of data over labeling.

Part II considers a superset of image recognition, in the sense that it interrogates AI models' ability to perform language-level understanding of the image content. Language,

much like vision, is highly compositional. In this part, I consider various choices to reflect this compositionality, via per-word visualization, program-like execution, or graph-like symbolic representation. Chapter 5 and Chapter 6 study per-word visualization for image captioning and referring expressions, respectively. Chapter 7 brings program-like execution into referring expression comprehension. Chapter 8 understands sentences as graphs, whose symbolic nature may be useful for tasks across computer vision and natural language processing.

Part III extends image recognition in the other direction. It explicitly takes into account the fact that the 2D images are generated from the 3D world, and I mainly study its implications for model testing and evaluation. Generating test cases from the 3D world is advantageous, because of clear physical meanings and better disentanglement. Chapter 9 explains how adversarial examples may be more concerning if generated from the 3D world, and shows their existence. Chapter 10 formalizes this idea as an evaluation protocol emphasizing the worst case instead of measuring the average case.

11.2 Future Directions

The two themes of this dissertation are automation (Part I) and diagnosis (Part II and Part III). I will discuss these two themes separately in terms of future directions that I think are important.

11.2.1 Automation

The field of AutoML has made big progress over the past few years. However, this appealing name can be misleading, and it should be acknowledged that existing efforts still belong to a *mixture* of human expertise and machine design. For example, the search space cannot be arbitrarily designed; it still needs to be carefully defined by human experts. A dream scenario would be to truly remove humans from machines' self-progression

towards AI, and hope this process can "start from scratch" and replicate key milestones, *e.g.*, (re-)discovering gradient descent, stacking layers, (re-)discovering backpropagation, utilizing invariance and equivariance (by tying weights), *etc*.

There are at least two challenges in realizing this dream scenario. The first challenge is: what is the "right amount" of information or knowledge we should give to the machines, in order to jump start this process? On the one hand, we want it to be "minimal" for the arguments made earlier. On the other hand, it seems that at the very least, we need to provide the machines with the equivalence of "incentive" or "curiosity" in order to start this self-progression and keep it going. The second challenge is: what can we do to make this process smart and fast? Nesting for loops to do blind enumeration does not seem to be a good solution. Evolution (used in an ambitious recent effort [209]) is better, as it is more sample-efficient while still being quite minimal. But can we do better in mimicking human's ability to quickly extract rules and attack a problem (whether known or unknown) in a principled manner?

11.2.2 Diagnosis

The motivation for doing diagnosis is to make sure that the model has "truly understood" the underlying concept, instead of learning something superficial. In some controlled environments, it may be possible to cover different combinations equally (*e.g.*, Chapter 7). However, this approach will inevitably run into two challenges. One, as more factors are incorporated, the number of possible scenarios will grow exponentially, and this approach can no longer scale. Two, in real world scenarios, we cannot control the distribution of input data, which is usually heavily biased as opposed to perfectly balanced. In fact, the latter can help explain the dilemma where in many applications, a model can achieve impressive numbers on the test set (*e.g.*, 99% accuracy), but especially in sensitive applications, we still would rather hand the job to a human with 98% accuracy on the test set.

I think the way to address this is to rethink and revamp the current evaluation protocol,

which measures the *average performance* on a *fixed test set* whose distribution is usually *identical to the training set*. In [285], my advisor and I argued a new paradigm along the line of measuring the *worst performance* on a *dynamic test set* whose distribution may *not necessarily be identical to the training set*. Chapter 10 is our first attempt to materialize this idea, but it is not perfect, and we need more efforts to make this idea more principled, more complete, and easier to implement. We notice that contemporary researchers [72], [122] are realizing this problem as well.

Finally, from the training perspective, a loss function emphasizing the worst case would have a better chance of learning the true compositionality (*i.e.*, getting the final 1% right), as opposed to superficial appearance matching (which may get you all the way to 99%). But of course, future experiments are needed to back this point up.

Appendix A

Progressive Neural Architecture Search

A.1 Search Efficiency of PNAS with RNN-ensemble

In Section 2.5.3 of the main chapter, we focused on the performance of MLP-ensemble as the surrogate model. Here we provide analysis of RNN-ensemble as well.

Again, each method is repeated 5 times to reduce the randomness in neural architecture search, and both performance mean and the variance are plotted in Figure A.1. A more quantitative breakdown is given in Table A.1. We see that PNAS with RNN-ensemble is

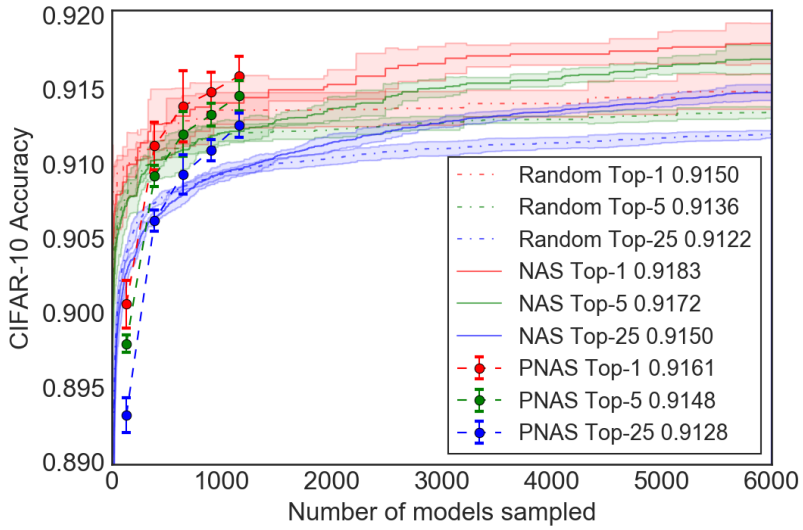


Figure A.1. Comparing the relative efficiency of PNAS (using RNN-ensemble) with NAS and random search under the same search space.

B	Top	Accuracy	# PNAS	# NAS	Speedup (# models)	Speedup (# examples)
5	1	0.9161	1160	2222	1.9	5.1
5	5	0.9148	1160	2489	2.1	5.4
5	25	0.9128	1160	2886	2.5	5.7

Table A.1. Relative efficiency of PNAS (using RNN-ensemble predictor) and NAS under the same search space.

about twice as efficient than NAS in terms of number of models trained and evaluated, and five times as efficient by the number of examples. Speedup measured by number of examples is greater than speedup in terms of number of models, because NAS has an additional reranking stage, that trains the top 250 models for 300 epochs each before picking the best one.

A.2 Searching Cells with More Blocks

Using the MLP-ensemble predictor, we tried to continue the progressive search beyond cells with 5 blocks, all the way till $B = 10$. The result of this experiment is visualized in Figure A.2, which extends Figure 2.4 of the main chapter. As can be seen, PNAS is able to find good performing models over the much larger search spaces of $B > 5$. Note that the unconstrained search space size increases by about 4 orders of magnitude for every B level, reaching $\sim 10^{33}$ possible model configurations at $B = 10$. This is one of the main advantages of PNAS, to examine a highly focused search space of arbitrary size progressively. Notice that the NAS curve for comparison is still for $B = 5$, and if we search cells with more blocks using NAS, this curve is likely to go down, because of the growth in search space.

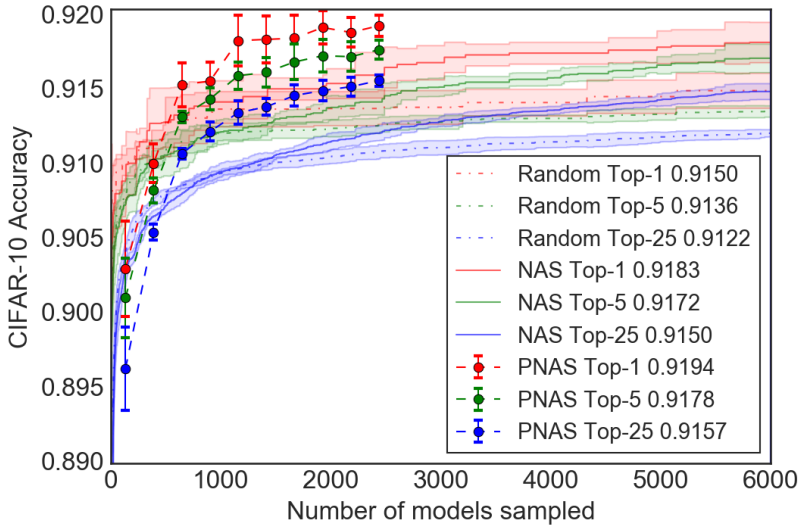


Figure A.2. Running PNAS (using MLP-ensemble) from cells with 1 block to cells with 10 blocks.

A.3 Intermediate Level PNASNet Models

Our Progressive Neural Architecture Search algorithm explores cells from simple to complex by growing the number of blocks. We choose $B = 5$, and indeed the best model found in the final level (PNASNet-5; visualized in the left plot of Figure 2.1) demonstrates state-of-the-art performance. In this section, however, we are interested in the best models found in smaller, intermediate levels, namely $b = 1, 2, 3, 4$. We call these models PNASNet-{1, 2, 3, 4}.

We visualize their cell structures in Figure A.3, and report their performances on CIFAR-10 in Table A.2. We see that the test set error rate decreases as we progress from $b = 1$ to $b = 5$, and the performances of these PNASNets with smaller number of blocks are still competitive.

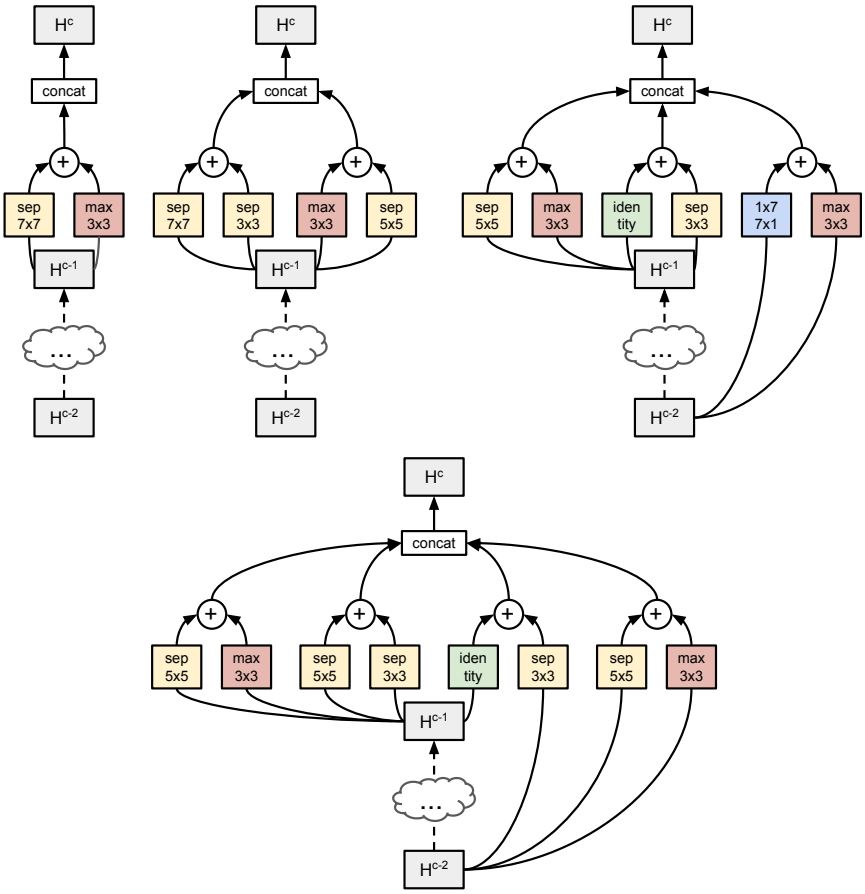


Figure A.3. Cell structures used in PNASNet-{1, 2, 3, 4}.

Model	B	N	F	Error	Params	M_1	E_1	M_2	E_2	Cost
PNASNet-4	4	4	44	3.50±0.10	3.0M	904	0.9M	0	0	0.8B
PNASNet-3	3	6	32	3.70±0.12	1.8M	648	0.9M	0	0	0.6B
PNASNet-2	2	6	32	3.73±0.09	1.7M	392	0.9M	0	0	0.4B
PNASNet-1	1	6	44	4.01±0.11	1.6M	136	0.9M	0	0	0.2B

Table A.2. Image classification performance on CIFAR test set. "Error" is the top-1 misclassification rate on the CIFAR-10 test set. (Error rates have the form $\mu \pm \sigma$, where μ is the average over multiple trials and σ is the standard deviation. In PNAS we use 15 trials.) "Params" is the number of model parameters. "Cost" is the total number of examples processed through SGD ($M_1 E_1 + M_2 E_2$) before the architecture search terminates.

A.4 Transferring from CIFAR-10 to ImageNet

Figure A.4 shows that the accuracy on CIFAR-10 (even for models which are only trained for 20 epochs) is strongly correlated with the accuracy on ImageNet, which proves that searching for models using CIFAR-10 accuracy as a fast proxy for ImageNet accuracy is a reasonable thing to do.

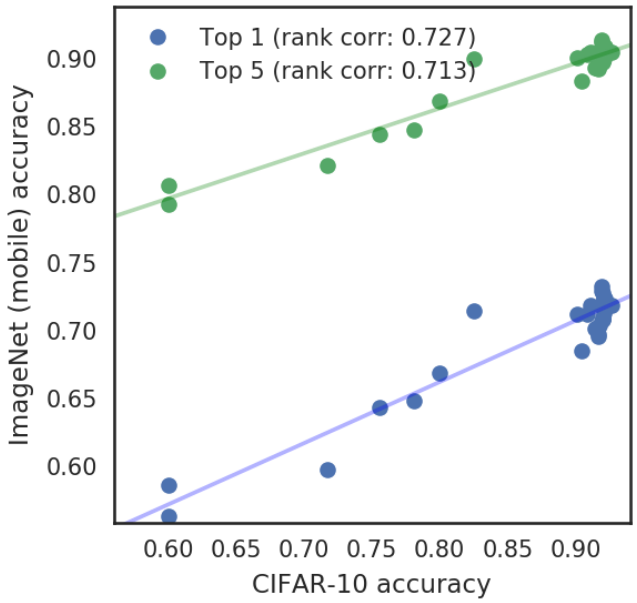


Figure A.4. Relationship between performance on CIFAR-10 and ImageNet for different neural network architectures. The high rank correlation of 0.727 (top-1) suggests that the best architecture searched on CIFAR-10 is general and transferable to other datasets. (Note, however, that rank correlation for the higher-value points (with CIFAR score above 0.89) is a bit lower: 0.505 for top-1, and 0.460 for top-5.)

Appendix B

Are Labels Necessary for Neural Architecture Search?

B.1 Additional Details for Search-Based Experiments

B.1.1 Search Phase

Recall that in the search phase of NAS-DARTS/UnNAS-DARTS, we consider 3 datasets: ImageNet-1K (IN1K), ImageNet-22K (IN22K), Cityscapes.

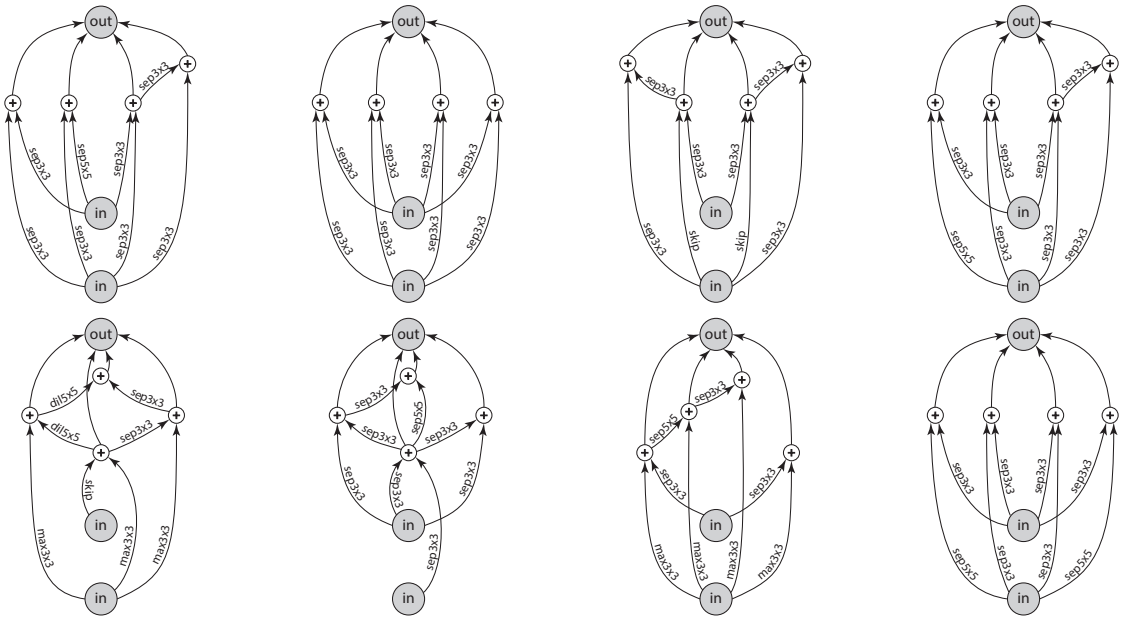
For IN1K, we follow [35], [150] to postpone updating architecture parameters α for half of the total search epochs. For IN1K/IN22K, the search phase lasts 2 epochs for IN1K and 1 epoch for IN22K. Since IN22K is approximately 10 times larger than IN1K (~14M vs ~1.2M), the search on IN22K is approximately 5 times longer than the search on IN1K. Batch size is 64, learning rate is 0.1 (cosine schedule), weight decay is 0.00003. For Cityscapes, the search phase lasts 400 epochs. Batch size is 32, learning rate is 0.1 (cosine schedule), weight decay is 0.0003. Other than those listed here and in the main chapter, all hyperparameters follow those used in the original DARTS. In a few settings where the batch size above will exceed 32GB GPU memory, we divide batch size and learning rate (both for weights and for architecture parameters α) by 2.

B.1.2 Evaluation Phase

When evaluating an architecture on Cityscapes semantic segmentation, since the task is pixel-level classification instead of image-level classification, we need to make minimal but necessary modifications. Different from the network used in IN1K classification, we (1) replace the last stride 2 layer with stride 1, to increase spatial resolution; and (2) remove the global average pooling and replace the fully connected classifier with the Atrous Spatial Pyramid Pooling (ASPP) module [31] followed by bilinear upsampling to produce per-pixel classification at the original resolution. These are the same modifications the segmentation framework DeepLabv3 [31] made to a ResNet backbone.

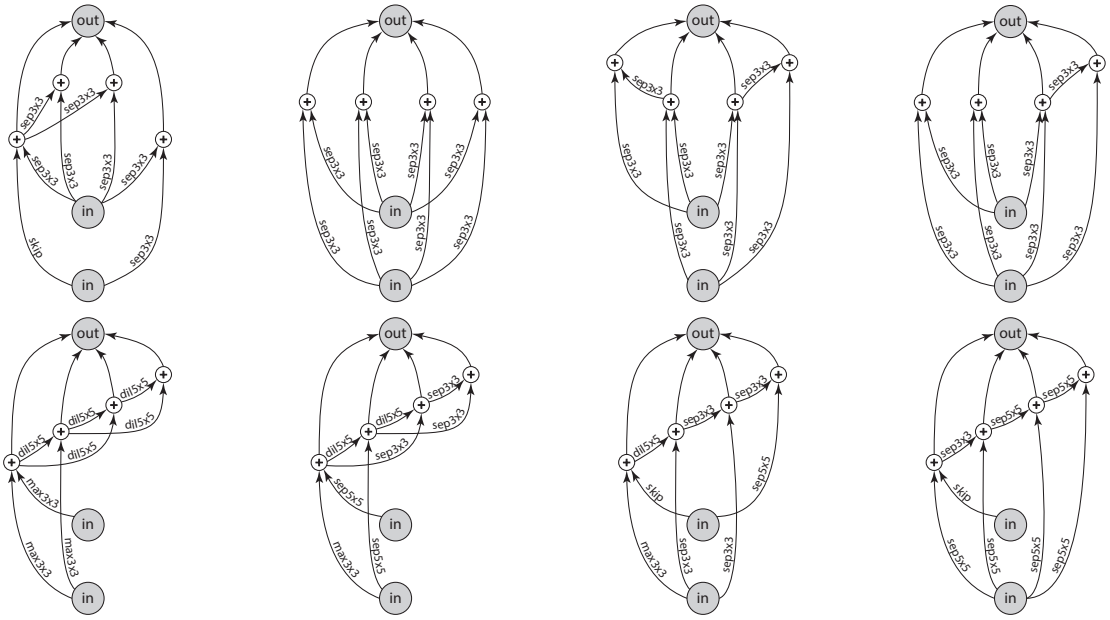
B.2 NAS-DARTS and UnNAS-DARTS Architectures

Here we visualize all the NAS-DARTS and UnNAS-DARTS cell architectures: searched on ImageNet-1K (Figure B.1), ImageNet-22K (Figure B.2), Cityscapes (Figure B.3).



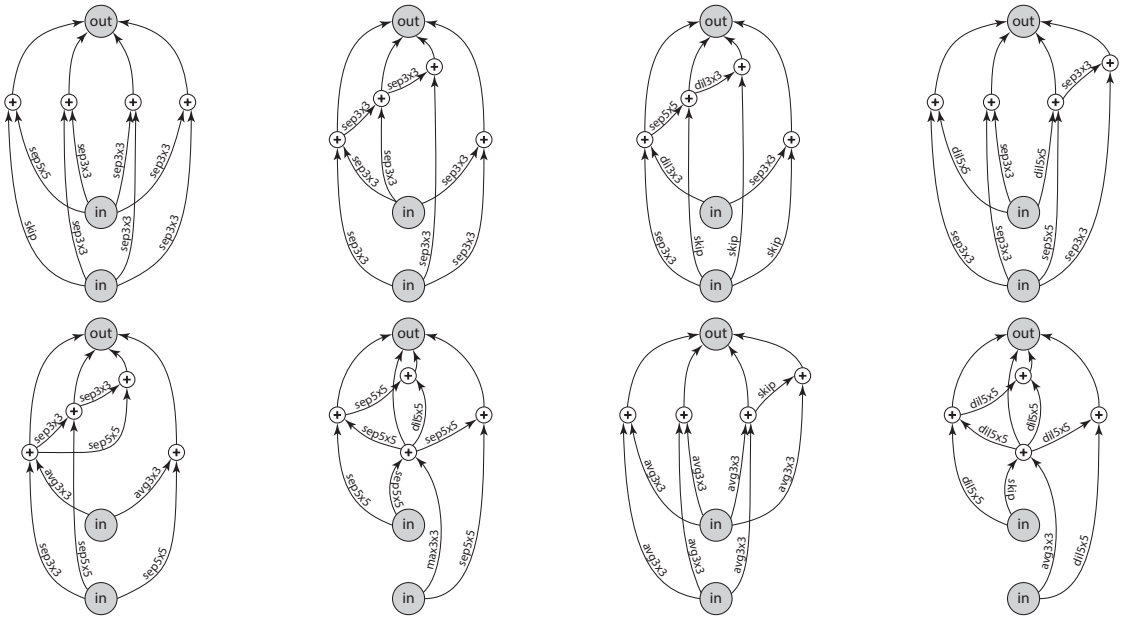
(a) NAS Supv.Cls **(b)** UnNAS Rot **(c)** UnNAS Color **(d)** UnNAS Jigsaw

Figure B.1. Cell architectures (normal and reduce) searched on ImageNet-1K.



(a) NAS `Supv.Cls` **(b)** UnNAS `Rot` **(c)** UnNAS `Color` **(d)** UnNAS `Jigsaw`

Figure B.2. Cell architectures (normal and reduce) searched on ImageNet-22K.



(a) NAS `Supv.Seg` **(b)** UnNAS `Rot` **(c)** UnNAS `Color` **(d)** UnNAS `Jigsaw`

Figure B.3. Cell architectures (normal and reduce) searched on Cityscapes.

Appendix C

CLEVR-Ref+: Diagnosing Visual Reasoning with Referring Expressions

In this appendix, we begin by providing network architecture details of IEP-Ref to supplement Section 7.4.1 of the main chapter. We then provide more analysis of the four models' performance on CLEVR-Ref+, to supplement Section 7.4.2 of the main chapter. Finally, we show more qualitative examples (referring expression and ground truth box/mask) from CLEVR-Ref+.

C.1 Network Architectures in IEP-Ref

In Figure 7.7 of the main chapter, we listed all modules used in our IEP-Ref model (except `Segment`). In IEP-Ref, each of these modules is parameterized with a small fully convolutional network and belongs to one of the following 4 categories:

- **Preprocess**: This component maps the image to the feature tensor. Its output is the input to the `Scene` module. See Table C.1 for the network architecture.
- **Unary**: This includes the `Scene`, `Filter_X`, `Unique`, `Relate`, `Same_X` modules. It transforms one feature tensor to another. See Table C.2 for the network architecture.
- **Binary**: This includes the `And` and `Or` modules. It transforms two feature tensors to one. See Table C.3 for the network architecture.

Layer	Output size
Input image	$3 \times 320 \times 320$
ResNet101 [89] conv4_6	$1024 \times 20 \times 20$
Conv($3 \times 3, 1024 \rightarrow 128$)	$128 \times 20 \times 20$
ReLU	$128 \times 20 \times 20$
Conv($3 \times 3, 128 \rightarrow 128$)	$128 \times 20 \times 20$
ReLU	$128 \times 20 \times 20$

Table C.1. Network architecture for the **Preprocess** module.

Index	Layer	Output size
(1)	Previous module output	$128 \times 20 \times 20$
(2)	Conv($3 \times 3, 128 \rightarrow 128$)	$128 \times 20 \times 20$
(3)	ReLU	$128 \times 20 \times 20$
(4)	Conv($3 \times 3, 128 \rightarrow 128$)	$128 \times 20 \times 20$
(5)	Residual: Add (1) and (4)	$128 \times 20 \times 20$
(6)	ReLU	$128 \times 20 \times 20$

Table C.2. Network architecture for the **Unary** modules.

- **Postprocess**: This only includes the `Segment` module. It transforms the 128-channel feature tensor to a 1-channel segmentation mask. See Table C.4 for the network architecture.

Network architectures for **Preprocess**, **Unary**, **Binary** are directly inherited from IEP [113].

C.2 More Model Analysis on CLEVR-Ref+

C.2.1 Number of Objects in a Scene

We suspect that the more objects in a scene, the harder for the model to carry out the referring reasoning steps. In Figure C.1 we plot the performance of each model with respect to the number of objects in a scene. All models drop in performance when the number of objects increases, suggesting that the models tend to struggle when dealing with too many objects.

Index	Layer	Output size
(1)	Previous module output	$128 \times 20 \times 20$
(2)	Previous module output	$128 \times 20 \times 20$
(3)	Concatenate (1) and (2)	$256 \times 20 \times 20$
(4)	Conv($1 \times 1, 256 \to 128$)	$128 \times 20 \times 20$
(5)	ReLU	$128 \times 20 \times 20$
(6)	Conv($3 \times 3, 128 \to 128$)	$128 \times 20 \times 20$
(7)	ReLU	$128 \times 20 \times 20$
(8)	Conv($3 \times 3, 128 \to 128$)	$128 \times 20 \times 20$
(9)	Residual: Add (5) and (8)	$128 \times 20 \times 20$
(10)	ReLU	$128 \times 20 \times 20$

Table C.3. Network architecture for the **Binary** modules.

Layer	Output size
Previous module output	$128 \times 20 \times 20$
Unary module	$128 \times 20 \times 20$
Conv($1 \times 1, 128 \to 128$)	$128 \times 20 \times 20$
ReLU	$128 \times 20 \times 20$
Bilinear upsample	$128 \times 320 \times 320$
Conv($1 \times 1, 128 \to 128$)	$128 \times 320 \times 320$
ReLU	$128 \times 320 \times 320$
Conv($1 \times 1, 128 \to 32$)	$32 \times 320 \times 320$
ReLU	$32 \times 320 \times 320$
Conv($1 \times 1, 32 \to 4$)	$4 \times 320 \times 320$
ReLU	$4 \times 320 \times 320$
Conv($1 \times 1, 4 \to 1$)	$1 \times 320 \times 320$

Table C.4. Network architecture for the `Segment` module.

C.2.2 Schedule of Acquiring Reasoning Abilities

We are interested to see if throughout the training process, the network exhibit a schedule of acquiring various reasoning abilities (e.g. spatial reasoning, logic etc). From Figure C.2, it seems that no such schedule was developed, and performance steadily increase across different referring expression categories. This may be due to the random sampling during training, instead of active learning (c.f. [181]).

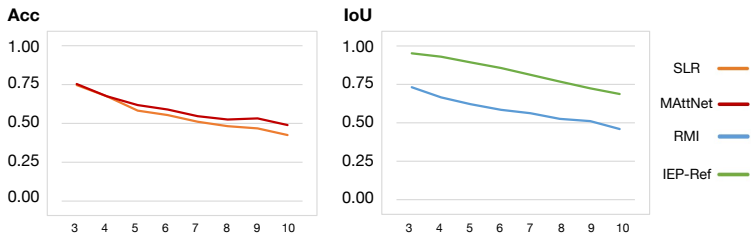


Figure C.1. Effect of number of objects in a scene on referring detection or segmentation performance.

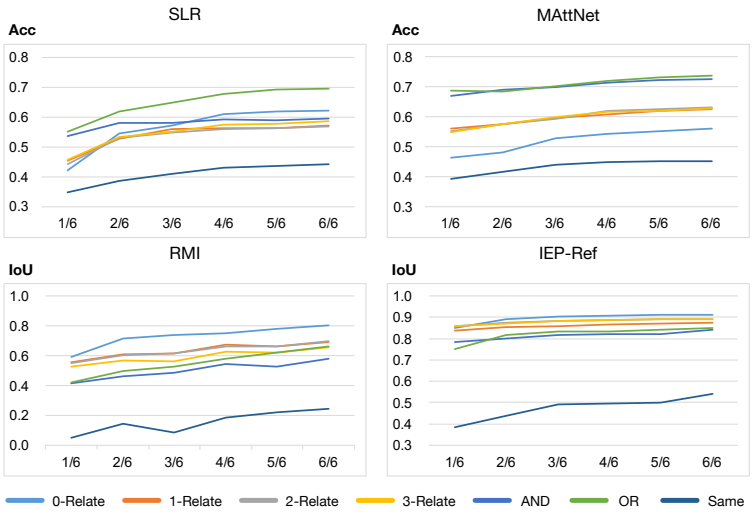


Figure C.2. Performance across different referring expression categories throughout training. We inspect the performance every 1/6 of the entire training iterations.

C.2.3 Novel Compositions

To further test the models' generalization ability, we also conducted experiments on the Compositional Generalization Test (CoGenT) data provided by CLEVR [112]. Here models are trained on objects with only a subset of all combinations, and then tested on both the same subset of combinations (*valA*) and another subset of combinations (*valB*). Results are summarized in Figure C.3. We see a very small gap for detection models, suggesting that they have learned compositionality to generalize well. The gap for segmentation models, on the other hand, is larger.

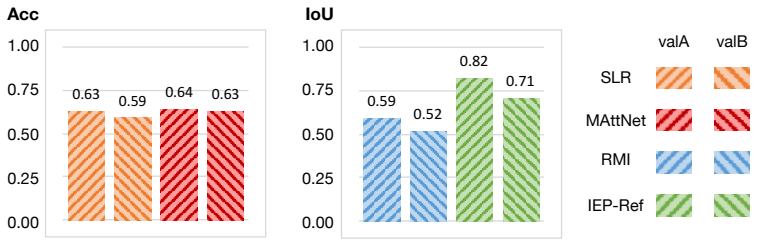
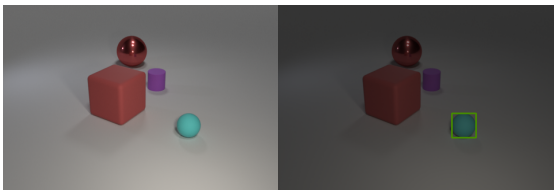


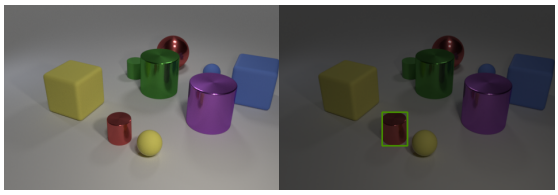
Figure C.3. Different models' performance on *valA* and *valB* of the CLEVR CoGenT data.

C.3 More Data Examples from CLEVR-Ref+

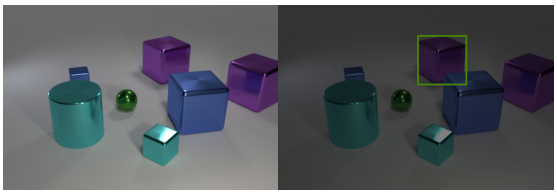
The remaining pages show random images, referring expressions, and the referring ground truth from our CLEVR-Ref+ dataset. In particular, we choose at least one example from each referring expression category (the 7 middle columns in Table 7.3 of the main chapter). We show both detection ground truth (Figure C.4) and segmentation ground truth (Figure C.5).



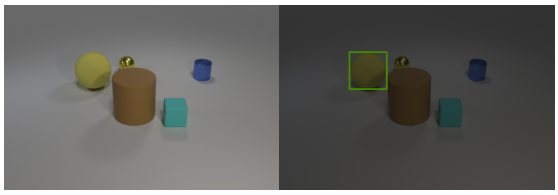
(a) *Look at matte thing that is on the left side of the red object that is behind the second one of the object(s) from right; The first one of the rubber thing(s) from front that are right of it*



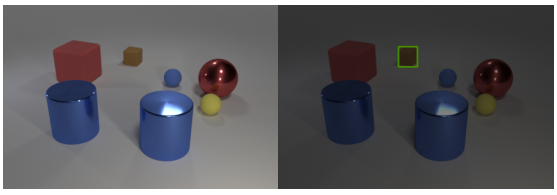
(b) *The objects that are the seventh one of the thing(s) from right that are in front of the nineth one of the thing(s) from front or the second one of the thing(s) from front*



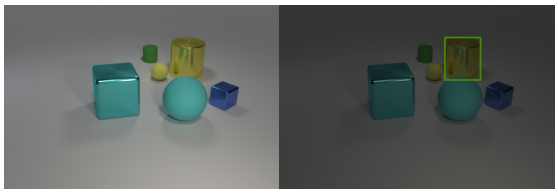
(c) *The big metallic object(s) that are both to the left of the third one of the large thing(s) from left and on the right side of the first one of the object(s) from front*



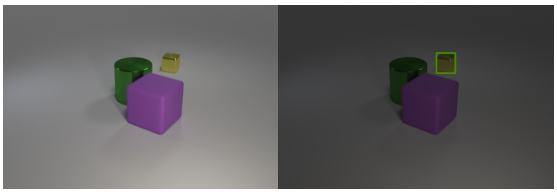
(d) *The fully visible yellow ball(s)*



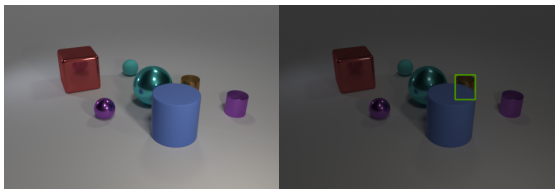
(e) *Any other things that are the same shape as the fourth one of the rubber thing(s) from right*



(f) *Find object that is behind the fifth one of the object(s) from left; The cylinder(s) that are to the right of it*

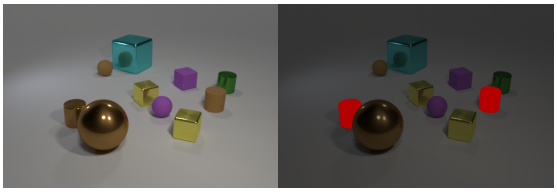


(g) *Look at partially visible object(s); The second one of the thing(s) from left that are on the right side of it*

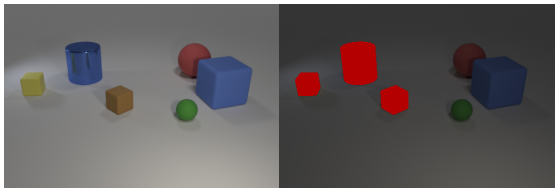


(h) *The second one of the shiny cylinder(s) from right that are to the right of the thing that is behind the thing that is on the left side of the first one of the tiny thing(s) from left*

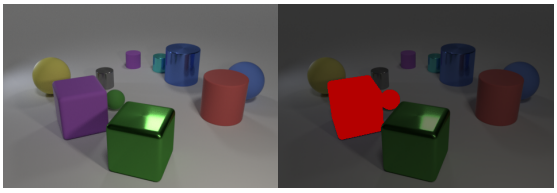
Figure C.4. Referring object detection examples from CLEVR-Ref+.



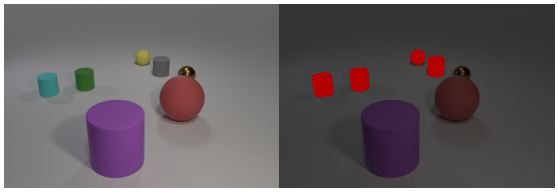
(a) *Any other things that are the same shape as the seventh one of the object(s) from front*



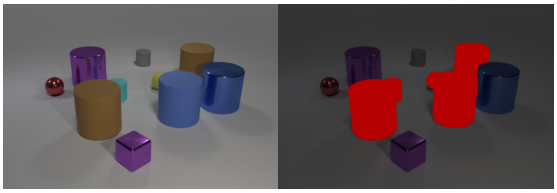
(b) *Look at rubber ball that is to the left of the red ball(s); The thing(s) that are left of it*



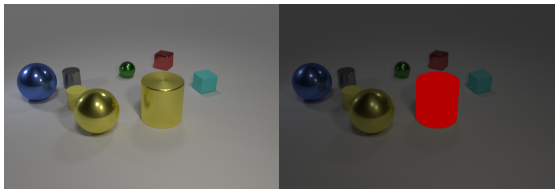
(c) *The rubber object(s) that are to the right of the sixth one of the rubber thing(s) from right and to the left of the fifth one of the object(s) from left*



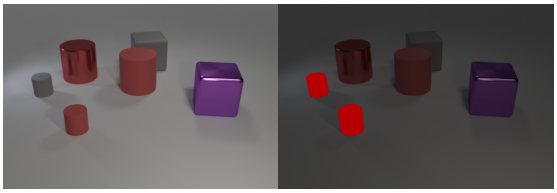
(d) *The fully visible small thing(s)*



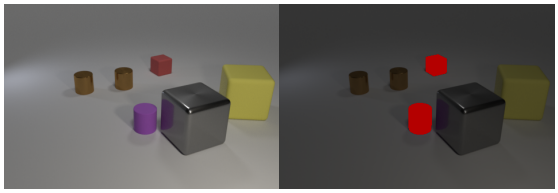
(e) *Look at tiny rubber cylinder that is behind the tiny object that is on the right side of the seventh one of the cylinder(s) from front; The rubber thing(s) that are in front of it*



(f) *The big things that are the sixth one of the object(s) from left or the seventh one of the object(s) from right*



(g) *Find the second one of the red rubber thing(s) from left; The fully visible rubber cylinder(s) that are in front of it*



(h) *Any other tiny object(s) made of the same material as the second one of the cube(s) from front*

Figure C.5. Referring image segmentation examples from CLEVR-Ref+.

Appendix D

Adversarial Attacks Beyond the Image Space

D.1 Attack Curves with Different (Differentiable or Non-Differentiable) Renderers

In Figure D.1, we plot how the average loss function value (probability of the original class, after softmax) changes with respect to the number of attack iterations. Image-space attacks often succeed very quickly, whereas physical-space attacks are much slower yet more difficult, especially for the factors of illumination and material.

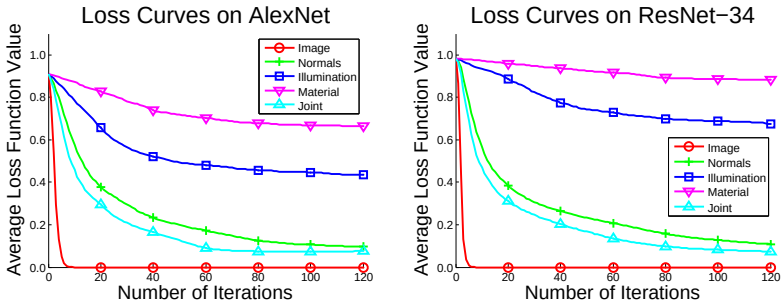


Figure D.1. Attack curves for 3D object classification with a differentiable renderer.

In Figure D.2, we plot how the average log-probability advantage (red) and image-space Euclidean distance (blue) change with respect to the number of attack iterations. An average log-probability advantage of 0 means that all images have been attacked

successfully. Physical-space attacks are much more difficult to succeed and also require a much larger perceptibility.

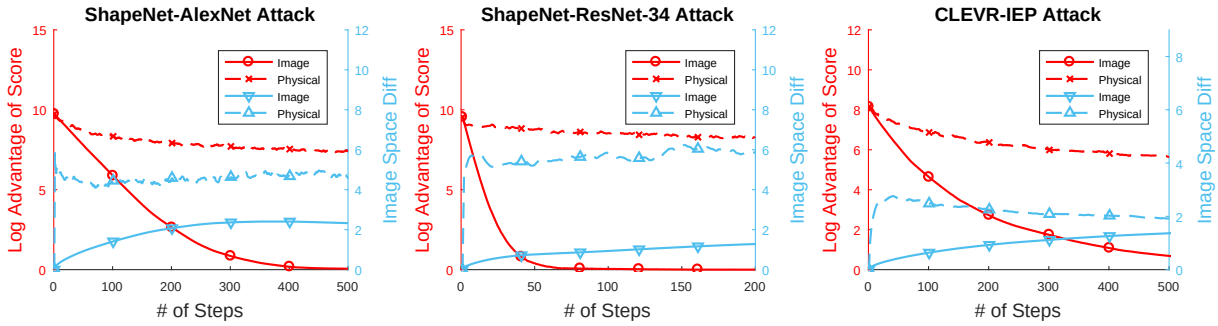


Figure D.2. Attack curves for 3D object classification and visual question answering with a non-differentiable renderer.

From these curves, we can conclude that physical-space attacks especially adding factors with clear physical meanings are much more difficult. This is arguably because most of these attacks impact the values of more than one pixels in the image space, which raises higher difficulties to the optimizers (*e.g.*, gradient-descent-based). We should also note that, with a more powerful optimizer, it is possible to find more adversarial examples in the physical world.

Bibliography

[1] K. Ahmed and L. Torresani, "Maskconnect: Connectivity learning by gradient descent," in *European Conference on Computer Vision*, 2018.

[2] M. A. Alcorn, Q. Li, Z. Gong, C. Wang, L. Mai, W.-S. Ku, and A. Nguyen, "Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects," in *Conference on Computer Vision and Pattern Recognition*, 2019.

[3] P. Anderson, B. Fernando, M. Johnson, and S. Gould, "SPICE: semantic propositional image caption evaluation," in *European Conference on Computer Vision*, 2016.

[4] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein, "Neural module networks," in *Conference on Computer Vision and Pattern Recognition*, 2016.

[5] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh, "VQA: visual question answering," in *International Conference on Computer Vision*, 2015.

[6] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: circumventing defenses to adversarial examples," in *International Conference on Machine Learning*, 2018.

[7] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok, "Synthesizing robust adversarial examples," in *International Conference on Machine Learning*, 2018.

[8] A. Azulay and Y. Weiss, "Why do deep convolutional networks generalize so poorly to small image transformations?," 2018. arXiv: 1805.12177.

[9] J. Ba, V. Mnih, and K. Kavukcuoglu, "Multiple object recognition with visual attention," in *International Conference on Learning Representations*, 2015.

[10] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *International Conference on Learning Representations*, 2015.

[11] A. Baisero, F. T. Pokorny, and C. H. Ek, "On a family of decomposable kernels on sequences," 2015. arXiv: 1501.06284.

[12] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *International Conference on Learning Representations*, 2017.

[13] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," 2017. arXiv: 1705.10823.

[14] S. Banerjee and A. Lavie, "Meteor: An automatic metric for mt evaluation with improved correlation with human judgments," in *Proceedings of the Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, 2005.

[15] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[16] Blender Online Community, *Blender – a 3d modelling and rendering package*, `https://www.blender.org/`, Blender Foundation, Blender Institute, Amsterdam, 2017.

[17] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SMASH: one-shot model architecture search through hypernetworks," in *International Conference on Learning Representations*, 2018.

[18] T. B. Brown, N. Carlini, C. Zhang, C. Olsson, P. Christiano, and I. Goodfellow, "Unrestricted adversarial examples," 2018. arXiv: `1809.08352`.

[19] S. R. Bulò, L. Porzi, and P. Kontschieder, "In-place activated batchnorm for memory-optimized training of dnns," in *Conference on Computer Vision and Pattern Recognition*, 2018.

[20] J. Buys and P. Blunsom, "Robust incremental neural semantic graph parsing," in *Annual Meeting of the Association for Computational Linguistics*, 2017.

[21] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *AAAI Conference on Artificial Intelligence*, 2018.

[22] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," in *International Conference on Learning Representations*, 2019.

[23] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *IEEE Symposium on Security and Privacy*, 2017.

[24] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, *et al.*, "Shapenet: an information-rich 3d model repository," 2015. arXiv: `1512.03012`.

[25] A. X. Chang, M. Savva, and C. D. Manning, "Learning spatial knowledge for text to 3d scene generation," in *Empirical Methods in Natural Language Processing*, 2014.

[26] D. Chen and C. D. Manning, "A fast and accurate dependency parser using neural networks," in *Empirical Methods in Natural Language Processing*, 2014.

[27] K. Chen, J. Wang, L.-C. Chen, H. Gao, W. Xu, and R. Nevatia, "Abc-cnn: An attention based convolutional neural network for visual question answering," 2015. arXiv: `1511.05960`.

[28] L.-C. Chen, M. D. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, and J. Shlens, "Searching for efficient multi-scale architectures for dense image prediction," in *Advances in Neural Information Processing Systems*, 2018.

[29] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," in *International Conference on Learning Representations*, 2015.

[30] ——, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[31] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," 2017. arXiv: 1706.05587.

[32] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *European Conference on Computer Vision*, 2018.

[33] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 2017.

[34] Q. Chen, W. Qiu, Y. Zhang, L. Xie, and A. Yuille, "Sampleahead: Online classifier-sampler communication for learning from synthesized data," in *British Machine Vision Conference*, 2018.

[35] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *International Conference on Computer Vision*, 2019.

[36] X. Chen and C. L. Zitnick, "Learning a recurrent visual representation for image caption generation," 2014. arXiv: 1411.5654.

[37] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng, "Dual path networks," in *Advances in Neural Information Processing Systems*, 2017.

[38] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Empirical Methods in Natural Language Processing*, 2014.

[39] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[40] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese, "3d-r2n2: A unified approach for single and multi-view 3d object reconstruction," in *European Conference on Computer Vision*, 2016.

[41] V. Cirik, L. Morency, and T. Berg-Kirkpatrick, "Visual referring expression recognition: What do systems actually learn?" In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2018.

[42] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Conference on Computer Vision and Pattern Recognition*, 2016.

[43] C. Cortes, X. Gonzalvo, V. Kuznetsov, M. Mohri, and S. Yang, "Adanet: Adaptive structural learning of artificial neural networks," in *International Conference on Machine Learning*, 2017.

[44] J. Cross and L. Huang, "Incremental parsing with minimal features using bi-directional LSTM," in *Annual Meeting of the Association for Computational Linguistics*, 2016.

[45] B. Dai, Y. Zhang, and D. Lin, "Detecting visual relationships with deep relational networks," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[46] A. Das, H. Agrawal, C. L. Zitnick, D. Parikh, and D. Batra, "Human attention in visual question answering: Do humans and deep networks look at the same regions?" In *Empirical Methods in Natural Language Processing*, 2016.

[47] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Conference on Computer Vision and Pattern Recognition*, 2009.

[48] M. J. Denkowski and A. Lavie, "Meteor universal: Language specific translation evaluation for any target language," in *Proceedings of the Ninth Workshop on Statistical Machine Translation*, 2014.

[49] T. Devries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," 2017. arXiv: 1708.04552.

[50] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *International Conference on Computer Vision*, 2015.

[51] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *International Joint Conference on Artificial Intelligence*, 2015.

[52] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *International Conference on Machine Learning*, 2014.

[53] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *Conference on Computer Vision and Pattern Recognition*, 2015.

[54] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun, "Ppp-net: Platform-aware progressive search for pareto-optimal neural architectures," in *International Conference on Learning Representations, Workshop Track*, 2018.

[55] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox, "Discriminative unsupervised feature learning with convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2014.

[56] T. Dozat and C. D. Manning, "Deep biaffine attention for neural dependency parsing," in *International Conference on Learning Representations*, 2017.

[57] C. Dyer, M. Ballesteros, W. Ling, A. Matthews, and N. A. Smith, "Transition-based dependency parsing with stack long short-term memory," in *Annual Meeting of the Association for Computational Linguistics*, 2015.

[58] T. Elsken, J. H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," 2017. arXiv: 1711.04528.

[59] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, "A rotation and a translation suffice: fooling cnns with simple transformations," 2017. arXiv: 1712.02779.

[60] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, 2015.

[61] M. Everingham, L. J. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman, "The pascal visual object classes (VOC) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.

[62] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning models," in *Conference on Computer Vision and Pattern Recognition*, 2018.

[63] H. Fang, S. Gupta, F. Iandola, R. K. Srivastava, L. Deng, P. Dollár, J. Gao, X. He, M. Mitchell, J. C. Platt, *et al.*, "From captions to visual concepts and back," in *Conference on Computer Vision and Pattern Recognition*, 2015.

[64] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1915–1929, 2012.

[65] C. Finn, I. J. Goodfellow, and S. Levine, "Unsupervised learning for physical interaction through video prediction," in *Advances in Neural Information Processing Systems*, 2016.

[66] J. Flanigan, S. Thomson, J. G. Carbonell, C. Dyer, and N. A. Smith, "A discriminative graph-based parser for the abstract meaning representation," in *Annual Meeting of the Association for Computational Linguistics*, 2014.

[67] D. Fourure, R. Emonet, E. Fromont, D. Muselet, A. Tremeau, and C. Wolf, "Residual conv-deconv grid network for semantic segmentation," in *British Machine Vision Conference*, 2017.

[68] A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, M. Ranzato, and T. Mikolov, "Devise: A deep visual-semantic embedding model," in *Advances in Neural Information Processing Systems*, 2013.

[69] A. Fukui, D. H. Park, D. Yang, A. Rohrbach, T. Darrell, and M. Rohrbach, "Multimodal compact bilinear pooling for visual question answering and visual grounding," in *Empirical Methods in Natural Language Processing*, 2016.

[70] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig, "Virtual worlds as proxy for multi-object tracking analysis," in *Conference on Computer Vision and Pattern Recognition*, 2016.

[71] H. Gao, J. Mao, J. Zhou, Z. Huang, L. Wang, and W. Xu, "Are you talking to a machine? dataset and methods for multilingual image question," in *Advances in Neural Information Processing Systems*, 2015.

[72] M. Gardner, Y. Artzi, V. Basmova, J. Berant, B. Bogin, S. Chen, P. Dasigi, D. Dua, Y. Elazar, A. Gottumukkala, *et al.*, "Evaluating nlp models via contrast sets," 2020. arXiv: 2004.02709.

[73] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "Dropblock: A regularization method for convolutional networks," in *Advances in Neural Information Processing Systems*, 2018.

[74] ——, "Nas-fpn: Learning scalable feature pyramid architecture for object detection," in *Conference on Computer Vision and Pattern Recognition*, 2019.

[75] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," in *International Conference on Learning Representations*, 2018.

[76] J. Gilmer, R. P. Adams, I. Goodfellow, D. Andersen, and G. E. Dahl, "Motivating the rules of the game for adversarial example research," 2018. arXiv: 1807.06732.

[77] A. Giusti, D. Ciresan, J. Masci, L. Gambardella, and J. Schmidhuber, "Fast image scanning with deep max-pooling convolutional neural networks," in *International Conference on Image Processing*, 2013.

[78] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations*, 2015.

[79] *Google automl vision api tutorial*, Nov. 2019. [Online]. Available: https://cloud.google.com/vision/automl/docs/tutorial.

[80] Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, and D. Parikh, "Making the V in VQA matter: Elevating the role of image understanding in visual question answering," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[81] K. Grauman and T. Darrell, "The pyramid match kernel: Discriminative classification with sets of image features," in *International Conference on Computer Vision*, 2005.

[82] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," 2015. arXiv: 1503.04069.

[83] R. Grosse, R. R. Salakhutdinov, W. T. Freeman, and J. B. Tenenbaum, "Exploiting compositionality to explore a large space of model structures," in *Conference on Uncertainty in Artificial Intelligence*, 2012.

[84] K. Gu, B. Yang, J. Ngiam, Q. Le, and J. Shlens, "Using videos to evaluate image model robustness," 2019. arXiv: 1904.10076.

[85] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji, and J. Malik, "Semantic contours from inverse detectors," in *International Conference on Computer Vision*, 2011.

[86] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Conference on Computer Vision and Pattern Recognition*, 2020.

[87] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *European Conference on Computer Vision*, 2014.

[88] ——, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *International Conference on Computer Vision*, 2015.

[89] ——, "Deep residual learning for image recognition," in *Conference on Computer Vision and Pattern Recognition*, 2016.

[90] D. Hershcovich, O. Abend, and A. Rappoport, "A transition-based directed acyclic graph parser for UCCA," in *Annual Meeting of the Association for Computational Linguistics*, 2017.

[91] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[92] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[93] M. Hodosh, P. Young, and J. Hockenmaier, "Framing image description as a ranking task: Data, models and evaluation metrics," *Journal of Artificial Intelligence Research*, vol. 47, pp. 853–899, 2013.

[94] M. Holschneider, R. Kronland-Martinet, J. Morlet, and P. Tchamitchian, "A real-time algorithm for signal analysis with the help of the wavelet transform," in *Wavelets: Time-Frequency Methods and Phase Space*, Springer, 1989, pp. 289–297.

[95] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017. arXiv: 1704.04861.

[96] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Conference on Computer Vision and Pattern Recognition*, 2018.

[97] R. Hu, J. Andreas, T. Darrell, and K. Saenko, "Explainable neural computation via stack neural module networks," in *European Conference on Computer Vision*, 2018.

[98] R. Hu, J. Andreas, M. Rohrbach, T. Darrell, and K. Saenko, "Learning to reason: End-to-end module networks for visual question answering," in *International Conference on Computer Vision*, 2017.

[99] R. Hu, M. Rohrbach, J. Andreas, T. Darrell, and K. Saenko, "Modeling relationships in referential expressions with compositional modular networks," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[100] R. Hu, M. Rohrbach, and T. Darrell, "Segmentation from natural language expressions," in *European Conference on Computer Vision*, 2016.

[101] R. Hu, M. Rohrbach, S. Venugopalan, and T. Darrell, "Utilizing large scale vision and text datasets for image segmentation from referring expressions," 2016. arXiv: 1608.08305.

[102] R. Hu, H. Xu, M. Rohrbach, J. Feng, K. Saenko, and T. Darrell, "Natural language object retrieval," in *Conference on Computer Vision and Pattern Recognition*, 2016.

[103] F. Huang, J. T. Ash, J. Langford, and R. E. Schapire, "Learning deep resnet blocks sequentially using boosting theory," in *International Conference on Machine Learning*, 2018.

[104] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[105] P. J. Huber, *Robust statistics*. Springer, 2011.

[106] D. A. Hudson and C. D. Manning, "Compositional attention networks for machine reasoning," in *International Conference on Learning Representations*, 2018.

[107] D. A. Hudson and C. D. Manning, "Gqa: A new dataset for real-world visual reasoning and compositional question answering," in *Conference on Computer Vision and Pattern Recognition*, 2019.

[108] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International Conference on Learning and Intelligent Optimization*, 2011.

[109] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning*. Springer, 2019.

[110] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, "Black-box adversarial attacks with limited queries and information," in *International Conference on Machine Learning*, 2018.

[111] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015.

[112] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. B. Girshick, "CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[113] J. Johnson, B. Hariharan, L. van der Maaten, J. Hoffman, L. Fei-Fei, C. L. Zitnick, and R. B. Girshick, "Inferring and executing programs for visual reasoning," in *International Conference on Computer Vision*, 2017.

[114] J. Johnson, R. Krishna, M. Stark, L. Li, D. A. Shamma, M. S. Bernstein, and F. Li, "Image retrieval using scene graphs," in *Conference on Computer Vision and Pattern Recognition*, 2015.

[115] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *International Conference on Machine Learning*, 2015.

[116] A. Kae, K. Sohn, H. Lee, and E. Learned-Miller, "Augmenting crfs with boltzmann machine shape priors for image labeling," in *Conference on Computer Vision and Pattern Recognition*, 2013.

[117] L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit, "One model to learn them all," 2017. arXiv: 1706.05137.

[118] A. Kanezaki, Y. Matsushita, and Y. Nishida, "Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints," in *Conference on Computer Vision and Pattern Recognition*, 2018.

[119] A. Kar, A. Prakash, M.-Y. Liu, E. Cameracci, J. Yuan, M. Rusiniak, D. Acuna, A. Torralba, and S. Fidler, "Meta-sim: Learning to generate synthetic datasets," in *International Conference on Computer Vision*, 2019.

[120] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Conference on Computer Vision and Pattern Recognition*, 2015.

[121] A. Katiyar and C. Cardie, "Going out on a limb: Joint extraction of entity mentions and relations without dependency trees," in *Annual Meeting of the Association for Computational Linguistics*, 2017.

[122] D. Kaushik, E. Hovy, and Z. C. Lipton, "Learning the difference that makes a difference with counterfactually-augmented data," in *International Conference on Learning Representations*, 2019.

[123] S. Kazemzadeh, V. Ordonez, M. Matten, and T. L. Berg, "Referitgame: Referring to objects in photographs of natural scenes," in *Empirical Methods in Natural Language Processing*, 2014.

[124] J. Kim, S. Lee, D. Kwak, M. Heo, J. Kim, J. Ha, and B. Zhang, "Multimodal residual learning for visual QA," in *Advances in Neural Information Processing Systems*, 2016.

[125] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[126] E. Kiperwasser and Y. Goldberg, "Simple and accurate dependency parsing using bidirectional LSTM feature representations," *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 313–327, 2016.

[127] R. Kiros, R. Salakhutdinov, and R. S. Zemel, "Unifying visual-semantic embeddings with multimodal neural language models," 2014. arXiv: 1411.2539.

[128] I. Konstas, S. Iyer, M. Yatskar, Y. Choi, and L. Zettlemoyer, "Neural AMR: sequence-to-sequence models for parsing and generation," in *Annual Meeting of the Association for Computational Linguistics*, 2017.

[129] S. Kornblith, J. Shlens, and Q. V. Le, "Do better imagenet models transfer better?" In *Conference on Computer Vision and Pattern Recognition*, 2019.

[130] P. Krähenbühl and V. Koltun, "Efficient inference in fully connected crfs with gaussian edge potentials," in *Advances in Neural Information Processing Systems*, 2011.

[131] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L. Li, D. A. Shamma, M. S. Bernstein, and L. Fei-Fei, "Visual genome: Connecting language and vision using crowdsourced dense image annotations," *International Journal of Computer Vision*, vol. 123, no. 1, pp. 32–73, 2017.

[132] J. Krishnamurthy and T. Kollar, "Jointly learning to parse and perceive: Connecting natural language to the physical world," *Transactions of the Association for Computational Linguistics*, vol. 1, pp. 193–206, 2013.

[133] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Technical report, University of Toronto*, 2009.

[134] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012.

[135] S. Kübler, R. T. McDonald, and J. Nivre, *Dependency Parsing*, ser. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2009.

[136] J. Kuen, Z. Wang, and G. Wang, "Recurrent attentional networks for saliency detection," in *Conference on Computer Vision and Pattern Recognition*, 2016.

[137] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *International Conference on Learning Representations, Workshop Track*, 2017.

[138] ——, "Adversarial machine learning at scale," in *International Conference on Learning Representations*, 2017.

[139] G. Laput, M. Dontcheva, G. Wilensky, W. Chang, A. Agarwala, J. Linder, and E. Adar, "Pixeltone: A multimodal interface for image editing," in *Conference on Human Factors in Computing Systems*, 2013.

[140] G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-deep neural networks without residuals," in *International Conference on Learning Representations*, 2017.

[141] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Conference on Computer Vision and Pattern Recognition*, 2006.

[142] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[143] R. Li, K. Li, Y.-C. Kuo, M. Shu, X. Qi, X. Shen, and J. Jia, "Referring image segmentation via recurrent refinement networks," in *Conference on Computer Vision and Pattern Recognition*, 2018.

[144] Y. Li, W. Ouyang, B. Zhou, K. Wang, and X. Wang, "Scene graph generation from objects, phrases and caption regions," in *International Conference on Computer Vision*, 2017.

[145] D. Lin, Y. Ji, D. Lischinski, D. Cohen-Or, and H. Huang, "Multi-scale context intertwining for semantic segmentation," in *European Conference on Computer Vision*, 2018.

[146] G. Lin, A. Milan, C. Shen, and I. Reid, "Refinenet: Multi-path refinement networks with identity mappings for high-resolution semantic segmentation," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[147] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[148] T.-Y. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," in *European Conference on Computer Vision*, 2014.

[149] C. Liu, F. Sun, C. Wang, F. Wang, and A. L. Yuille, "MAT: A multimodal attentive translator for image captioning," in *International Joint Conference on Artificial Intelligence*, 2017.

[150] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei, "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," in *Conference on Computer Vision and Pattern Recognition*, 2019.

[151] C. Liu, P. Dollár, K. He, R. Girshick, A. Yuille, and S. Xie, "Are labels necessary for neural architecture search?" In *European Conference on Computer Vision*, 2020.

[152] C. Liu, Z. Lin, X. Shen, J. Yang, X. Lu, and A. L. Yuille, "Recurrent multimodal interaction for referring image segmentation," in *International Conference on Computer Vision*, 2017.

[153] C. Liu, J. Mao, F. Sha, and A. L. Yuille, "Attention correctness in neural image captioning," in *AAAI Conference on Artificial Intelligence*, 2017.

[154] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *European Conference on Computer Vision*, 2018.

[155] G. Liu, D. Ceylan, E. Yumer, J. Yang, and J. M. Lien, "Material editing using a physically based rendering network," in *International Conference on Computer Vision*, 2017.

[156] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *International Conference on Learning Representations*, 2018.

[157] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *International Conference on Learning Representations*, 2019.

[158] R. Liu, C. Liu, Y. Bai, and A. L. Yuille, "Clevr-ref+: Diagnosing visual reasoning with referring expressions," in *Conference on Computer Vision and Pattern Recognition*, 2019.

[159] W. Liu, A. Rabinovich, and A. C. Berg, "Parsenet: Looking wider to see better," 2015. arXiv: 1506.04579.

[160] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," in *International Conference on Learning Representations*, 2017.

[161] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Conference on Computer Vision and Pattern Recognition*, 2015.

[162] I. Loshchilov and F. Hutter, "SGDR: stochastic gradient descent with restarts," in *International Conference on Learning Representations*, 2017.

[163] C. Lu, R. Krishna, M. S. Bernstein, and F. Li, "Visual relationship detection with language priors," in *European Conference on Computer Vision*, 2016.

[164] R. Luo, F. Tian, T. Qin, and T.-Y. Liu, "Neural architecture optimization," in *Advances in Neural Information Processing Systems*, 2018.

[165] R. Luo and G. Shakhnarovich, "Comprehension-guided referring expressions," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[166] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Empirical Methods in Natural Language Processing*, 2015.

[167] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[168] M. Malinowski, M. Rohrbach, and M. Fritz, "Ask your neurons: A neural-based approach to answering questions about images," in *International Conference on Computer Vision*, 2015.

[169] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Annual Meeting of the Association for Computational Linguistics, System Demonstrations*, 2014.

[170] J. Mao, J. Huang, A. Toshev, O. Camburu, A. Yuille, and K. Murphy, "Generation and comprehension of unambiguous object descriptions," in *Conference on Computer Vision and Pattern Recognition*, 2016.

[171] J. Mao, W. Xu, Y. Yang, J. Wang, Z. Huang, and A. Yuille, "Deep captioning with multimodal recurrent neural networks (m-rnn)," in *International Conference on Learning Representations*, 2015.

[172] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of english: The penn treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[173] E. Margffoy-Tuay, J. C. Pérez, E. Botero, and P. Arbeláez, "Dynamic multimodal instance segmentation guided by natural language queries," in *European Conference on Computer Vision*, 2018.

[174] D. Mascharka, P. Tran, R. Soklaski, and A. Majumdar, "Transparency by design: Closing the gap between performance and interpretability in visual reasoning," in *Conference on Computer Vision and Pattern Recognition*, 2018.

[175] J. McCormac, A. Handa, S. Leutenegger, and A. Davison, "Scenenet rgb-d: 5m photorealistic images of synthetic indoor trajectories with ground truth," in *International Conference on Computer Vision*, 2017.

[176] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter, "Towards automatically-tuned neural networks," in *Workshop on Automatic Machine Learning*, 2016.

[177] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, "On detecting adversarial perturbations," in *International Conference on Learning Representations*, 2017.

[178] R. Miikkulainen, J. Z. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving deep neural networks," 2017. arXiv: 1703.00548.

[179] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013. arXiv: 1301.3781.

[180] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, 2013.

[181] I. Misra, R. B. Girshick, R. Fergus, M. Hebert, A. Gupta, and L. van der Maaten, "Learning by asking questions," in *Conference on Computer Vision and Pattern Recognition*, 2018.

[182] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, "Recurrent models of visual attention," in *Advances in Neural Information Processing Systems*, 2014.

[183] S. M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[184] S. M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Conference on Computer Vision and Pattern Recognition*, 2016.

[185] V. K. Nagaraja, V. I. Morariu, and L. S. Davis, "Modeling context between objects for referring expression understanding," in *European Conference on Computer Vision*, 2016.

[186] N. Narodytska and S. P. Kasiviswanathan, "Simple black-box adversarial perturbations for deep networks," 2016. arXiv: 1612.06299.

[187] R. Negrinho and G. J. Gordon, "Deeparchitect: Automatically designing and training deep architectures," 2017. arXiv: 1704.08792.

[188] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," in *European Conference on Computer Vision*, 2016.

[189] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: high confidence predictions for unrecognizable images," in *Conference on Computer Vision and Pattern Recognition*, 2015.

[190] F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis, "Geometrical considerations and nomenclature for reflectance," in *Radiometry*, 1992, pp. 94–145.

[191] K. Nishino, "Directional statistics brdf model," in *International Conference on Computer Vision*, 2009.

[192] J. Nivre, "Incrementality in deterministic dependency parsing," in *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, 2004.

[193] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *International Conference on Computer Vision*, 2015.

[194] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," in *European Conference on Computer Vision*, 2016.

[195] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," 2018. arXiv: 1807.03748.

[196] G. Papandreou, I. Kokkinos, and P.-A. Savalle, "Modeling local and global deformations in deep learning: Epitomic convolution, multiple instance learning, and sliding window detection," in *Conference on Computer Vision and Pattern Recognition*, 2015.

[197] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *IEEE Symposium on Security and Privacy*, 2016.

[198] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation," in *Annual Meeting of the Association for Computational Linguistics*, 2002.

[199] K. Pei, Y. Cao, J. Yang, and S. Jana, "Towards practical verification of machine learning: the case of computer vision systems," 2017. arXiv: 1712.01785.

[200] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing*, 2014.

[201] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. C. Courville, "Film: Visual reasoning with a general conditioning layer," in *AAAI Conference on Artificial Intelligence*, 2018.

[202] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *International Conference on Machine Learning*, 2018.

[203] B. A. Plummer, L. Wang, C. M. Cervantes, J. C. Caicedo, J. Hockenmaier, and S. Lazebnik, "Flickr30k entities: Collecting region-to-phrase correspondences for richer image-to-sentence models," in *International Conference on Computer Vision*, 2015.

[204] T. Pohlen, A. Hermans, M. Mathias, and B. Leibe, "Full-resolution residual networks for semantic segmentation in street scenes," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[205] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: deep learning on point sets for 3d classification and segmentation," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[206] I. Radosavovic, J. Johnson, S. Xie, W.-Y. Lo, and P. Dollár, "On network design spaces for visual recognition," in *International Conference on Computer Vision*, 2019.

[207] A. Ray, G. Christie, M. Bansal, D. Batra, and D. Parikh, "Question relevance in VQA: identifying non-visual and false-premise questions," in *Empirical Methods in Natural Language Processing*, 2016.

[208] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI Conference on Artificial Intelligence*, 2019.

[209] E. Real, C. Liang, D. R. So, and Q. V. Le, "Automl-zero: Evolving machine learning algorithms from scratch," 2020. arXiv: 2003.03384.

[210] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *International Conference on Machine Learning*, 2017.

[211] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[212] A. Rohrbach, M. Rohrbach, R. Hu, T. Darrell, and B. Schiele, "Grounding of textual phrases in images by reconstruction," in *European Conference on Computer Vision*, 2016.

[213] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2015.

[214] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[215] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Conference on Computer Vision and Pattern Recognition*, 2018.

[216] A. Santoro, D. Raposo, D. G. T. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap, "A simple neural network module for relational reasoning," in *Advances in Neural Information Processing Systems*, 2017.

[217] S. Saxena and J. Verbeek, "Convolutional neural fabrics," in *Advances in Neural Information Processing Systems*, 2016.

[218] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. arXiv: 1707.06347.

[219] S. Schuster, R. Krishna, A. Chang, L. Fei-Fei, and C. D. Manning, "Generating semantically precise scene graphs from textual descriptions for improved image retrieval," in *Proceedings of the Fourth Workshop on Vision and Language*, 2015.

[220] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," in *International Conference on Learning Representations*, 2014.

[221] K. Sfikas, T. Theoharis, and I. Pratikakis, "Exploiting the panorama representation for convolutional neural network classification and retrieval," in *Eurographics Workshop on 3D Object Retrieval*, 2017.

[222] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[223] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 640–651, 2017.

[224] T. Shi, L. Huang, and L. Lee, "Fast(er) exact decoding and global training for transition-based dependency parsing via a minimal feature set," in *Empirical Methods in Natural Language Processing*, 2017.

[225] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, and W. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," in *Advances in Neural Information Processing Systems*, 2015.

[226] K. J. Shih, S. Singh, and D. Hoiem, "Learning to localize little landmarks," in *Conference on Computer Vision and Pattern Recognition*, 2016.

[227] R. Shin, C. Packer, and D. Song, "Differentiable neural network architecture search," in *International Conference on Learning Representations, Workshop Track*, 2018.

[228] A. Shrivastava, R. Sukthankar, J. Malik, and A. Gupta, "Beyond skip connections: Top-down modulation for object detection," 2016. arXiv: 1612.06851.

[229] M. Shu, C. Liu, W. Qiu, and A. Yuille, "Identifying model weakness with adversarial examiner," in *AAAI Conference on Artificial Intelligence*, 2019.

[230] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[231] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[232] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[233] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, 2012.

[234] D. R. So, C. Liang, and Q. V. Le, "The evolved transformer," in *International Conference on Machine Learning*, 2019.

[235] Y. Song, R. Shu, N. Kushman, and S. Ermon, "Constructing unrestricted adversarial examples with generative models," in *Advances in Neural Information Processing Systems*, 2018.

[236] C. Spearman, "The proof and measurement of association between two things.," *The American Journal of Psychology*, 1904.

[237] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," in *International Conference on Machine Learning*, 2010.

[238] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[239] K. O. Stanley, *Neuroevolution: A different kind of deep learning*, Jul. 2017. [Online]. Available: https://www.oreilly.com/radar/neuroevolution-a-different-kind-of-deep-learning/.

[240] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[241] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *International Conference on Computer Vision*, 2015.

[242] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, 2014.

[243] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *International Conference on Learning Representations*, 2014.

[244] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning.," in *AAAI Conference on Artificial Intelligence*, 2017.

[245] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Conference on Computer Vision and Pattern Recognition*, 2015.

[246] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Conference on Computer Vision and Pattern Recognition*, 2016.

[247] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Conference on Computer Vision and Pattern Recognition*, 2019.

[248] D. Teney, L. Liu, and A. van den Hengel, "Graph-structured representations for visual question answering," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[249] Y. Tian, D. Krishnan, and P. Isola, "Contrastive multiview coding," 2019. arXiv: 1906.05849.

[250] F. Tramer, A. Kurakin, N. Papernot, D. Boneh, and P. McDaniel, "Ensemble adversarial training: attacks and defenses," in *International Conference on Learning Representations*, 2018.

[251] G. Underwood, L. Jebbett, and K. Roberts, "Inspecting pictures for information to verify a sentence: Eye movements in general encoding and in focused search," *Quarterly Journal of Experimental Psychology Section A*, vol. 57, no. 1, pp. 165–182, 2004.

[252] I. Vendrov, R. Kiros, S. Fidler, and R. Urtasun, "Order-embeddings of images and language," in *International Conference on Learning Representations*, 2016.

[253] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Conference on Computer Vision and Pattern Recognition*, 2015.

[254] C. Wang, N. Xue, and S. Pradhan, "A transition-based algorithm for AMR parsing," in *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015.

[255] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell, "Understanding convolution for semantic segmentation," in *Winter Conference on Applications of Computer Vision*, 2018.

[256] Y.-S. Wang, C. Liu, X. Zeng, and A. Yuille, "Scene graph parsing as dependency parsing," in *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2018.

[257] W. Wang, M. Yan, and C. Wu, "Multi-granularity hierarchical attention fusion networks for reading comprehension and question answering," in *Annual Meeting of the Association for Computational Linguistics*, 2018.

[258] X. Wang and A. Gupta, "Unsupervised learning of visual representations using videos," in *International Conference on Computer Vision*, 2015.

[259] K. Werling, G. Angeli, and C. D. Manning, "Robust subgraph generation improves abstract meaning representation parsing," in *Annual Meeting of the Association for Computational Linguistics*, 2015.

[260] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[261] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016. arXiv: 1609.08144.

[262] Y. Wu and K. He, "Group normalization," in *European Conference on Computer Vision*, 2018.

[263] Z. Wu, Y. Xiong, S. Yu, and D. Lin, "Unsupervised feature learning via non-parametric instance discrimination," in *Conference on Computer Vision and Pattern Recognition*, 2018.

[264] Z. Wu, C. Shen, and A. van den Hengel, "Wider or deeper: Revisiting the resnet model for visual recognition," 2016. arXiv: 1611.10080.

[265] Y. Xiang, R. Mottaghi, and S. Savarese, "Beyond pascal: A benchmark for 3d object detection in the wild," in *Winter Conference on Applications of Computer Vision*, 2014.

[266] T. Xiao, Y. Liu, B. Zhou, Y. Jiang, and J. Sun, "Unified perceptual parsing for scene understanding," in *European Conference on Computer Vision*, 2018.

[267] C. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie, and A. L. Yuille, "Adversarial examples for semantic segmentation and object detection," in *International Conference on Computer Vision*, 2017.

[268] L. Xie and A. L. Yuille, "Genetic CNN," in *International Conference on Computer Vision*, 2017.

[269] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[270] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei, "Scene graph generation by iterative message passing," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[271] H. Xu and K. Saenko, "Ask, attend and answer: Exploring question-guided spatial attention for visual question answering," in *European Conference on Computer Vision*, 2016.

[272] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International Conference on Machine Learning*, 2015.

[273] X. Xu, X. Chen, C. Liu, A. Rohrbach, T. Darell, and D. Song, "Can you fool ai with adversarial examples on a visual turing test?," 2017. arXiv: 1709.08693.

[274] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong, "Pc-darts: Partial channel connections for memory-efficient differentiable architecture search," in *International Conference on Learning Representations*, 2020.

[275] D. Yang, C. Xiao, B. Li, J. Deng, and M. Liu, "Realistic adversarial examples in 3d meshes," 2018. arXiv: 1810.05206.

[276] J. Yang, Z. Ren, M. Xu, X. Chen, D. Grandall, D. Parikh, and D. Batra, "Embodied visual recognition," 2019. arXiv: 1904.04404.

[277] Z. Yang, Y. Yuan, Y. Wu, W. W. Cohen, and R. Salakhutdinov, "Review networks for caption generation," in *Advances in Neural Information Processing Systems*, 2016.

[278] Z. Yang, X. He, J. Gao, L. Deng, and A. J. Smola, "Stacked attention networks for image question answering," in *Conference on Computer Vision and Pattern Recognition*, 2016.

[279] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "Nas-bench-101: Towards reproducible neural architecture search," in *International Conference on Machine Learning*, 2019.

[280] Q. You, H. Jin, Z. Wang, C. Fang, and J. Luo, "Image captioning with semantic attention," in *Conference on Computer Vision and Pattern Recognition*, 2016.

[281] P. Young, A. Lai, M. Hodosh, and J. Hockenmaier, "From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions," *Transactions of the Association for Computational Linguistics*, vol. 2, pp. 67–78, 2014.

[282] L. Yu, Z. Lin, X. Shen, J. Yang, X. Lu, M. Bansal, and T. L. Berg, "Mattnet: Modular attention network for referring expression comprehension," in *Conference on Computer Vision and Pattern Recognition*, 2018.

[283] L. Yu, P. Poirson, S. Yang, A. C. Berg, and T. L. Berg, "Modeling context in referring expressions," in *European Conference on Computer Vision*, 2016.

[284] L. Yu, H. Tan, M. Bansal, and T. L. Berg, "A joint speaker-listener-reinforcer model for referring expressions," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[285] A. L. Yuille and C. Liu, "Deep nets: What have they ever done for vision?," 2018. arXiv: 1805.04025.

[286] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *British Machine Vision Conference*, 2016.

[287] X. Zeng, C. Liu, Y.-S. Wang, W. Qiu, L. Xie, Y.-W. Tai, C.-K. Tang, and A. L. Yuille, "Adversarial attacks beyond the image space," in *Conference on Computer Vision and Pattern Recognition*, 2019.

[288] P. Zhang, Y. Goyal, D. Summers-Stay, D. Batra, and D. Parikh, "Yin and yang: Balancing and answering binary visual questions," in *Conference on Computer Vision and Pattern Recognition*, 2016.

[289] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *European Conference on Computer Vision*, 2016.

[290] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," 2017. arXiv: 1707.01083.

[291] X. Zhang, Z. Li, C. C. Loy, and D. Lin, "Polynet: A pursuit of structural diversity in very deep networks," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[292] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[293] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Conference on Computer Vision and Pattern Recognition*, 2018.

[294] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ade20k dataset," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[295] Y. Zhu, O. Groth, M. Bernstein, and L. Fei-Fei, "Visual7w: Grounded question answering in images," *Conference on Computer Vision and Pattern Recognition*, 2016.

[296] Y. Zhuang, F. Yang, L. Tao, C. Ma, Z. Zhang, Y. Li, H. Jia, X. Xie, and W. Gao, "Dense relation network: Learning consistent and context-aware representation for semantic image segmentation," in *International Conference on Image Processing*, 2018.

[297] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *International Conference on Learning Representations*, 2017.

[298] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Conference on Computer Vision and Pattern Recognition*, 2018.

Chenxi Liu

CONTACT INFORMATION

Department of Computer Science
Johns Hopkins University
Malone 260
3400 North Charles Street
Baltimore, Maryland 21218, USA

(312) 866-8175
cxliu@jhu.edu
https://cs.jhu.edu/~cxliu/

EDUCATION

Johns Hopkins University 2016 - 2020
Ph.D. in Computer Science
Advisor: Alan Yuille
Committee: Gregory Hager, Fei-Fei Li

University of California, Los Angeles 2015 - 2016
M.S. in Statistics
Advisor: Alan Yuille
GPA: 3.92

Rice University Fall 2013
Exchange Student in Electrical and Computer Engineering
GPA: 3.98

Tsinghua University 2011 - 2015
B.E. in Automation
Minor in Economics
GPA: 92/100 Ranking: 3/145

EXPERIENCE

Facebook AI Research, Menlo Park, CA Summer 2019
Research Intern
Mentors: Saining Xie, Ross Girshick, Piotr Dollár, Kaiming He

Google, Sunnyvale, CA Summer 2018
Software Engineering Intern
Mentors: Fei-Fei Li, Wei Hua, Liang-Chieh Chen

Google, Sunnyvale, CA Fall 2017
Software Engineering Intern
Mentors: Jia Li, Wei Hua, Jonathan Huang, Jonathon Shlens, Barret Zoph, Kevin Murphy, Fei-Fei Li

Adobe Research, San Jose, CA Summer 2016
Research Scientist Intern
Mentors: Zhe Lin, Xiaohui Shen, Jimei Yang, Xin Lu

Toyota Technological Institute at Chicago, Chicago, IL Summer 2015
Research Assistant
Advisors: Gregory Shakhnarovich, Michael Maire

University of Toronto, Toronto, ON Summer 2014
Research Assistant
Advisors: Raquel Urtasun, Sanja Fidler

TEACHING

Johns Hopkins University Fall 2018
Role: Teaching Assistant
Course: Probabilistic Models of the Visual Cortex
Instructor: Alan Yuille
Lectures: Bayes Decision Theory, Vision and Language

PUBLICATIONS

Chenxi Liu, Piotr Dollár, Kaiming He, Ross Girshick, Alan Yuille, and Saining Xie. 2020. Are Labels Necessary for Neural Architecture Search?. In *Proceedings of European Conference on Computer Vision*. Springer, Online. **(Spotlight)**

Michelle Shu, **Chenxi Liu**, Weichao Qiu, and Alan Yuille. 2020. Identifying Model Weakness with Adversarial Examiner. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, New York, New York, pages 11998-12006.

Siyuan Qiao, Huiyu Wang, **Chenxi Liu**, Wei Shen, and Alan Yuille. 2019. Rethinking Normalization and Elimination Singularity in Neural Networks. *CoRR, abs/1911.09738*.

Zhuotun Zhu, **Chenxi Liu**, Dong Yang, Alan Yuille, and Daguang Xu. 2019. V-NAS: Neural Architecture Search for Volumetric Medical Image Segmentation. In *Proceedings of the IEEE International Conference on 3D Vision (3DV)*. IEEE Computer Society, Quebec City, Canada, pages 240-248.

Siyuan Qiao, Huiyu Wang, **Chenxi Liu**, Wei Shen, and Alan Yuille. 2019. Weight Standardization. *CoRR, abs/1903.10520*.

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan Yuille, and Li Fei-Fei. 2019. Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Long Beach, California, pages 82-92. **(Oral)**

Xiaohui Zeng, **Chenxi Liu**, Yu-Siang Wang, Weichao Qiu, Lingxi Xie, Yu-Wing Tai, Chi Keung Tang, and Alan Yuille. 2019. Adversarial Attacks Beyond the Image Space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Long Beach, California, pages 4302-4311. **(Oral)**

Runtao Liu, **Chenxi Liu**, Yutong Bai, and Alan Yuille. 2019. CLEVR-Ref+: Diagnosing Visual Reasoning with Referring Expressions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Long Beach, California, pages 4185-4194.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive Neural Architecture Search. In *Proceedings of European Conference on Computer Vision*. Springer, Munich, Germany, pages 19-35. **(Oral)**

Alan Yuille and **Chenxi Liu**. 2018. Deep Nets: What have they ever done for Vision?. *Center for Brains, Minds and Machines (CBMM) Memo No. 088*.

Siyuan Qiao, **Chenxi Liu**, Wei Shen, and Alan Yuille. 2018. Few-Shot Image Recognition by Predicting Parameters from Activations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Salt Lake City, Utah, pages 7229-7238. **(Spotlight)**

Yu-Siang Wang, **Chenxi Liu**, Xiaohui Zeng, and Alan Yuille. 2018. Scene Graph Parsing as Dependency Parsing. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. Association for Computational Linguistics, New Orleans, Louisiana, pages 397-407. **(Oral)**

Chenxi Liu, Zhe Lin, Xiaohui Shen, Jimei Yang, Xin Lu, and Alan Yuille. 2017. Recurrent Multimodal Interaction for Referring Image Segmentation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE Computer Society, Venice, Italy, pages 1280-1289.

Yan Wang, Lingxi Xie, **Chenxi Liu**, Siyuan Qiao, Ya Zhang, Wenjun Zhang, Qi Tian, and Alan Yuille. 2017. SORT: Second-Order Response Transform for Visual Recognition. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE Computer Society, Venice, Italy, pages 1368-1377.

Siyuan Qiao, Wei Shen, Weichao Qiu, **Chenxi Liu**, and Alan Yuille. 2017. ScaleNet: Guiding Object Proposal Generation in Supermarkets and Beyond. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE Computer Society, Venice, Italy, pages 1809-1818.

Chenxi Liu, Junhua Mao, Fei Sha, and Alan Yuille. 2017. Attention Correctness in Neural Image Captioning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, San Francisco, California, pages 4176-4182.

Chenxi Liu*, Alexander Schwing*, Kaustav Kundu, Raquel Urtasun, and Sanja Fidler. 2015. Rent3D: Floor-Plan Priors for Monocular Layout Estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Boston, Massachusetts, pages 3413-3421. **(Oral)**

Jingwen Bai, **Chenxi Liu**, and Ashutosh Sabharwal. 2014. Increasing Cellular Capacity Using ISM Band Side-channels: A First Study. In *Proceedings of the 4th workshop on All things cellular: operations, applications, & challenges*. ACM, Chicago, Illinois, pages 9-14.

Awards

2019 Google PhD Fellowship	2019
Finalist, 2019 Adobe Research Fellowship	2018
Finalist, 2018 NVIDIA Graduate Fellowship	2018
Honorable Mention, 2017 Snap Research Fellowship	2017
Finalist, 2018 Adobe Research Fellowship	2017
Beijing Outstanding Graduate	2015
Tsinghua University Excellent Graduate	2015
National Southwest Associated University Scholarship, Tsinghua University	2014
Meritorious Winner, Mathematical Contest in Modeling, COMAP	2014
Geru Zheng Scholarship, 1st class, Tsinghua University	2013
National Scholarship, Tsinghua University	2013
3rd Prize, Beijing Universities Physics Competition	2012
EMC Scholarship, Tsinghua University	2012
Freshman Scholarship, 2nd class, Tsinghua University	2011

Invited Talks

Neural Architecture Search: Acceleration and Generalization

- Facebook — January 2020
- ByteDance — February 2020

- Amazon — February 2020
- Waymo — February 2020
- Facebook — March 2020

Progressive Neural Architecture Search
- Google Research Conference — October 2018
- Johns Hopkins University — October 2018
- Leiphone Webinar — September 2018
- European Conference on Computer Vision — September 2018
- Stanford University (Host: Fei-Fei Li, Amir Zamir) — July 2018
- Tsinghua University (Host: Jiwen Lu) — April 2018
- Valse Webinar (Host: Wei Shen) — December 2017

Rent3D: Floor-Plan Priors for Monocular Layout Estimation
- IEEE Conference on Computer Vision and Pattern Recognition — June 2015

Patents

Xin Lu, Zhe Lin, Xiaohui Shen, Jimei Yang, Jianming Zhang, Jen-Chan Jeff Chien, **Chenxi Liu**. Deep salient content neural networks for efficient digital object segmentation. *US Patent 10460214*.

Zhe Lin, Xin Lu, Xiaohui Shen, Jimei Yang, **Chenxi Liu**. Automatically segmenting images based on natural language phrases. *US Patent 10089742*.

Chenxi Liu, Tianlin Shi, Xinyi Yang. Method for identifying and correcting piano rhythms by using intelligent robot. *CN Patent 101930680*.

Service

Co-organizer of LVVU@CVPR 2020.

Invited reviewer for NIPS 2016, CVPR 2019, JMLR, NAACL 2019, PAMI, ICCV 2019, Neurocomputing, ACL 2019, EMNLP 2019, TMM, AAAI 2020, CVPR 2020, ACL 2020, ECCV 2020, NAS@ICLR 2020, EMNLP 2020, AACL-IJCNLP 2020, AutoML@ICML 2020, BMVC 2020, IJCV.

Reviewer for CVPR 2017, ICIP 2017, ICCV 2017, NIPS 2017, AAAI 2018, PRCV 2018, Pattern Recognition, AAAI 2019, ICML 2019.

Advising

Xiaohui Zeng (HKUST)	Now MSc student at U of T (Raquel Urtasun, Sanja Fidler)
Yu-Siang Wang (NTU)	Now MScAC student at U of T
Runtao Liu (PKU)	Now Ph.D. student at UC Berkeley (Stella Yu)
Michelle Shu (JHU)	Now Ph.D. student at Cornell University (Ramin Zabih)
Jiteng Mu (JHU)	Now Ph.D. student at UC San Diego (Xiaolong Wang)

Miscellaneous

Programming Languages	Python, Matlab, C++, C, LaTeX
Deep Learning Tools	PyTorch, Tensorflow, Caffe, Theano
TOEFL	117 (Reading 29, Listening 30, Speaking 29, Writing 29)
GRE	335 (Verbal 160, Quantitative 170, Writing 5.0)

References

Alan Yuille (`alan.l.yuille@gmail.com`), Professor, Johns Hopkins University
Kevin Murphy (`kpmurphy@google.com`), Research Scientist, Google AI
Fei-Fei Li (`feifeili@cs.stanford.edu`), Professor, Stanford University